# Energy-efficient Fine-grained Many-core Architecture for Video and DSP Applications

By

ZHIBIN XIAO
B.S. (Zhejiang University, Hangzhou, China), 2003
M.S. (Zhejiang University, Hangzhou, China), 2006

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

Davis

Approved:

_____
Chair, Dr. Bevan M. Baas

_____
Member, Dr. Venkatesh Akella

_____
Member, Dr. Soheil Ghiasi

Committee in Charge
2012

© Copyright by Zhibin Xiao 2012
All Rights Reserved

# Abstract

Many-core processor architecture has become the most promising computer architecture. However, how to utilize the extra system performance for real applications such as video encoding is still challenging. This dissertation investigates architecture design, physical implementation and performance evaluation of a fine-grained many-core processor for advanced video coding with a focus on interconnection, topology, memory system and related parallel programming methodology.

A baseline residual encoder for H.264/AVC on a current generation fine-grained many-core system is proposed that utilizes no application-specific hardware. The 25-processor encoder encodes video sequences with variable frame sizes and can encode 1080p HDTV at 30 frames per second with 293 mW average power consumption by adjusting each processor to workload-based optimal clock frequencies and dual supply voltages—a 38.4% power reduction compared to operation with only one clock frequency and supply voltage. In comparison to published implementations on the TI C642 DSP platform, the design has approximately 2.9–3.7 times higher scaled throughput, 11.2–15.0 times higher throughput per chip area, and 4.5–5.8 times lower energy per pixel. Compared to a heterogeneous SIMD architecture customized for H.264, the presented design has 2.8–3.6 times greater throughput, 4.5–5.9 times higher area efficiency, and similar energy efficiency.

Next, this dissertation proposes novel processor shapes and inter-connection topologies for many-core processor arrays which result in an overall application processor that requires fewer cores and has a lower total communication length. The proposed topologies compared to the commonly-used 2D mesh and include two 8-neighbor topologies, two 5-nearest-neighbor and three 6-nearest-neighbor topologies—three of which utilize 5-sided or hexagonal processor tiles. A 1080p H.264/AVC residual video encoder and a complete 54 Mbps 802.11a/11g wireless LAN baseband receiver are mapped onto all topologies and compared. The methodology to implement an array of hexagonal-shaped processor tiles with industry-standard CAD tools and automatic place and route flow is described. A 16-

bit DSP processor tile is tailored for all proposed topologies and implemented at 65 nm CMOS technology without full-custom layout. Results show that the 6-neighbor hexagonal tile and the 6-neighbor rectangular tile incur a 2.9% area increase per tile compared to the 4-neighbor 2D mesh, but their much more effective inter-processor interconnect yields an average total application area reduction of 21% and a total application inter-processor communication distance reduction of 19%.

Motivated by the fact that video encoding tasks normally read and write a block of data at one time in one transaction, the third part of this dissertation proposes a novel source synchronous bufferless shared memory to enable safe memory sharing among multiple processors with different clock domains. Compared with the previous FIFO buffered memory design, the bufferless memory module achieves lower latency, higher throughput, lower area overhead and lower power consumption. The bufferless memory module also supports direct communication with far-away processors through the existing processor-processor circuit switch interconnection network. The implementation results show that a 16 KB bufferless memory module reduces 58% single memory access latency and has higher burst-mode throughput (1%) compared to the 16 KB buffered memory module. The bufferless memory module also reduces the area overhead from 63% to 17% compared with buffered memory module, which yields a power reduction by 43%.

# Acknowledgments

Completing the PhD study is probably one of the most challenging tasks in my life. When I look back, I would never forget the days and nights I have spent with my fellow colleagues in the VCL lab at the Department of Electrical and Computer Engineering. The PhD study is a long journey and finally it comes to an end. Now I would like to thank all of the individuals who made this mission possible.

My first debt of gratitude must go to my advisor, Dr. Bevan M. Baas. He patiently provided the vision, encouragement and advice necessary for me to proceed through the doctorial program and complete my dissertation. I have benefited so much from his guidance on critical thinking, clear writing and effective presentation. The full financial support is also invaluable in the past economic environment. His devotion and enthusiasm on computer engineering research will continue to influence me in a positive way in my future career.

I also want to thank my qualification committee and dissertation reading committee members including professor Venkatesh Akella, professor Soheil Ghiasi, professor Zhaojun Bai, professor Rajeevan Amirtharajah for their useful comments and valuable feedbacks on my research.

My special thanks go to Dean Truong and Anh Tran, my fellow colleagues and good friends. We have spent most of the time in the lab together. I really appreciate their selfless help on my work and I will miss our discussions on all aspects of research. Being together with them in both study and leisure time is a great pleasure to me.

I would also like to thank previous VCL video group members, Stephen Le and Henna Huang. Their help and contribution on the video encoding project are essential to make this research presentable.

I want to thank previous VCL group members including Zhiyi Yu, Tinoosh Mohsenin, Toney Jacobson, Eric Work, Wayne Cheng and Paul Vincent Mejia. It was with them that I had a happy time when I first came to Davis. It is my pleasure to work with them.

I also would like to thank previous and current VCL lab members: Bin, Brent, Aaron, Jon, Jeremy, Emmanuel, Michael, Samir, Houshmand, Nima, Trevin, Lucas. I have enjoyed many discussions with them on various topics and found that there is always something I can learn from them.

Specially, I want to express my deep appreciation to my wife Shuting. Her constant support has allowed me to spend days and nights on this dissertation. She also brought us the most wonderful gift, our lovely daughter Amelia.

I want to thank my parents, my sister, my relatives and all of my friends. You might not know the details of my research area, but the support and help I get from you all might be more important than the pure academic help. It is because of you that I am a happy person and can keep pursuing my dreams.

Finally, I want to gratefully acknowledge supports from ST Microelectronics, Intel Corporation, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, SRC GRC Grant 1598, CSR Grant 1659, Intellasys Corporation, S Machines and the support of the C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Due to advances in images and video algorithms as well as very large scale integration (VLSI) technology, diverse and interesting visual experiences have been brought to our daily life. A number of international standards have contributed to the great success of image and video coding applications such as 3D high-definition TV (3D HDTV), online video streaming and portable video players (PVPs) [1]. The state-of-art image and video coding applications present challenges from the perspective of both hardware and software for embedded systems. These applications involve complex media processing tasks which have predictable execution behaviors and high computational and memory bandwidth requirements. Traditionally, there are several design approaches for multimedia applications such as application-specific integrated circuits (ASICs), programmable digital signal processors (DSPs) and field programmable gate arrays (FPGAs). ASICs can offer the highest performance and energy efficiency, but they have little programming flexibility. On the other hand, programmable DSPs are easy to program but their performance and energy efficiency is normally 10–100 times lower than ASICs. FPGAs fall in between the above two approaches.

An ideal platform for embedded multimedia applications should offer high computational performance, high energy efficiency and a high degree of flexibility. The flexibility

is a necessity to achieve high system integration in the presence of multiple standards and to support the diverse and rapidly evolving multimedia applications. The emerging *multi-core* or *many-core* systems provide us an opportunity to achieve this goal. Normally tiled architectures that integrate two or more independent processor cores are called multi-core processors. Manufacturers typically integrate multi-core processors into a single integrated circuit die (known as chip multiprocessors or CMP). CMPs that integrate tens, hundreds, or thousands of cores per die are called *many-core* chips.

## 1.1 Challenges

In the past 40 years, multimedia systems evolve with the rapid development of VLSI technology. More and more complicated image and video algorithms become feasible by upgrading the underlying hardware using newer process technology. The realtime software approach of advanced video encoding are possible with the performance boost of microprocessors and introduction of multimedia extension instructions. However, two major challenges are driving the current change towards multi-core and many-core in processor design.

The first challenge is the so-called "power wall" problem imposed by increasing circuit frequency and transistor density. Before the year of 2000, extensive research has been conducted to increase the performance of single processors by deepening the pipeline, increasing clock rate and decreasing transistor size which raises the power density to an unacceptable level. Figure 1.1 shows the power consumption of main Intel microprocessors from 1970 to 2008. Borkar has found that the power consumption of Intel processors follows Moore's law increasing from 0.3 W to 100 W from 1970 to 2000 [2]. The power density also increased from a couple of watts per $mm^2$ to about 100 Watts per $mm^2$. Although Intel tried to solve the power wall problem, the recent high-end Intel processors (from 2009 to 2012) still consume 100 to 130 Watts which is the maximum without incurring large costs

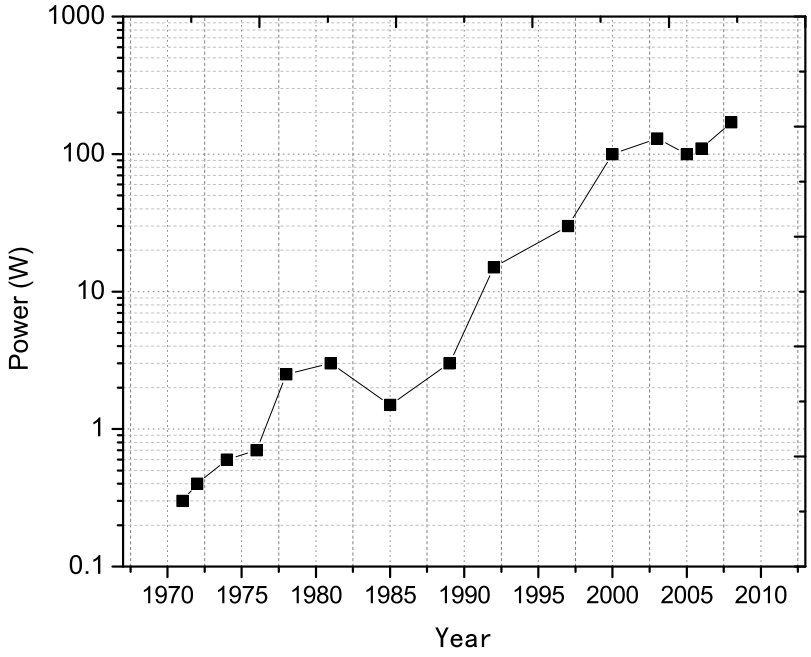



Figure 1.1: Power consumption of Intel microprocessor from 1970 to 2008; data from 1970 to 2000 are from [2]; data in the year of from 2003 to 2008 are from [6], [7], [8], [9] respectively.

for cooling [3–5].

In order to resolve the power wall problem and keep increasing the performance, the current trend of microprocessor design is to simplify each individual processor core while increasing the number of cores on the same chip. By putting multi-cores in the same chip, computer architects can keep improving transistor size and throughput of the processor without increasing power densities. Individual processor performance decreases while the number of processors increases means that more parallelism from applications should be exploited to distribute more tasks to more cores while reducing the amount of work on each core.

The second major challenge is the so-called “memory wall” problem imposed by multi-core systems. According to Hennessy and Patterson, processor performance has increased 55% each year since 1980, while memory performance increase by only 7% each year [10]. For multi-core systems, this performance gap between processors and memories has widened significantly, because more processors are integrated into the chip while the number of chip

pins for memory is limited. Algorithms now need to be more aware of the memory bandwidth and latency. Efficient communication mechanisms are more important than before. It is critical to optimize the memory subsystem to minimize off-chip memory bandwidth subject to the constraint of available on-chip memory for high throughput and data intensive multimedia applications. Generally, this can be accomplished by transforming and optimizing the algorithm memory access pattern.

In summary, multi-core and many-core processors have become the most promising computer architecture for next-generation computation. The design and implementation of an efficient multi-core or many-core processor is challenging. Furthermore, although parallel systems guarantee continuing increase of system performance, how to utilize the extra system performance for real applications like advanced video coding is also challenging. Parallelizing algorithms is not an easy task. Serious challenges must be resolved in order to implement advanced video encoding on a many-core system. This dissertation focuses on designing and applying a massively-parallel fine-grained many-core architecture for advanced video encoding.

## 1.2 Contributions

This dissertation makes a couple of contributions.

- It proposes a fine-grained parallel programming methodology and successfully demonstrates that fine-grained many-core architecture can achieve high performance and energy efficiency for both video encoding algorithms with high data-level parallelism like integer transform and quantization and serial algorithms with fine-grained task-level parallelism like CAVLC. The proposed programming methodology yields an H.264/AVC residual encoder capable of realtime 1080p (1920x1080) HDTV encoding with both higher energy efficiency and area efficiency compared with other software approaches in common DSPs and customized hybrid multi-core architec-

tures. In comparison to published implementations on the TI C642 DSP platform, the design has approximately 2.9–3.7 times higher scaled throughput, 11.2–15.0 times higher throughput per chip area, and 4.5–5.8 times lower energy per pixel. Compared to a heterogeneous SIMD architecture customized for H.264, the presented design has 2.8–3.6 times greater throughput, 4.5–5.9 times higher area efficiency, and similar energy efficiency.

- It proposes novel processor shapes and inter-connection topologies for many-core processor arrays which result in an overall application processor that requires fewer cores and has a lower total communication length. The proposed topologies compared to the commonly-used 2D mesh and include two 8-neighbor topologies, two 5-nearest-neighbor and three 6-nearest-neighbor topologies—three of which utilize 5-sided or hexagonal processor tiles. A 1080p H.264/AVC residual video encoder and a 54 Mbps 802.11a/g OFDM wireless LAN baseband receiver are mapped onto all topologies. The 6-neighbor hexagonal tile incurs a 2.9% area increase per tile compared to the 4-neighbor 2D mesh, but its much more effective inter-processor interconnect yields an average total application area reduction of 22% and an average application power savings of 17%.

- It demonstrates the feasibility of using commonly available commercial CAD tools to implement tiled CMPs with all of the proposed topologies. All processor tiles were designed using a standard cell flow up to the layout-level just before GDS extraction. The implementation results justify the proposed topologies which have small area overhead and little performance and energy penalties while providing much more effective inter-processor interconnect to reduce application area and communication link lengths.

- It proposes a novel source synchronous bufferless shared memory to enable safe memory sharing among multiple processors with different clock domains. Compared

with the previous FIFO buffered memory design, the bufferless memory achieves lower latency, higher throughput, lower area overhead and lower power consumption. The bufferless memory also supports direct communication with far-away processors through the existing processor-processor circuit switch interconnection network. The implementation results show that a 16 KB bufferless memory module reduces 58% single memory access latency and has slightly higher throughput (1%) in a burst mode compared to the 16 KB buffered memory module. The bufferless memory module also reduces the area overhead from 63% to 17% compared with buffered memory module, which yields a power reduction by 43%.

## 1.3 Organization

This dissertation is organized as follows. After the introduction, chapter 2 gives an overview of fine-grained many-core architecture for video encoding. Chapter 3 analyzes H.264/AVC encoding algorithms in terms of computation complexity and memory requirement and discusses the parallelization methods. Chapter 4 proposes a high-performance H.264/AVC baseline residual encoder for current many-core system. A thorough performance analysis and comparison is given. Chapter 5 proposes novel processor shapes and topologies and demonstrates their effectiveness by real-word application mapping and physical implementation. Chapter 6 investigates the distributed shared memory systems for fine-grained many-core architecture with detailed physical implementation and performance evaluation. Chapter 7 presents conclusions and future work.

# Chapter 2

# Background and Research Goals

Many-core systems provide both opportunity and challenges for advanced video coding. In this chapter, section 2.1 first introduces the research goals. The general features of basic video encoding algorithms and parallel programming approach are described. The current and proposed fine-grained many-core system architecture is introduced. All of the proposed features help to address the challenges described in chapter 1. In section 2.2, a survey of related parallel architectures for multimedia applications is given. The difference between the proposed architecture and existing architectures is also highlighted.

## 2.1 Research Goals

The goal of this research is to explore the capability of advanced video coding on a fine-grained many-core system like the asynchronous array of simple processors (AsAP) architecture, which comprises a 2-D array of reduced complexity programmable processors with small memories interconnected by a reconfigurable mesh network [11]. This multiprocessor architecture efficiently makes use of task-level parallelism in many complex DSP applications, and also efficiently computes many large tasks using fine-grained parallelism.

The research goal can be divided into two separate perspectives. The first part is studying, parallelizing and optimizing the video encoding algorithms on the current AsAP sys-

tem. The second part of this research focuses on application-driven architecture design and implementation of the next generation AsAP processor. The following subsections illustrate the project goal specifically.

### 2.1.1 Multimedia Application Characteristics and Approaches

Researchers have studied the characteristics of multimedia applications in the past decades [12–16]. According to the past researches, multimedia applications are long believed to exhibit the following features:

- Video algorithms typically repeat a small set of operations over a continuous data set. The intensive computation for highly regular operations shows high data parallelism.
- Multimedia applications always operate on a narrow data types. an 8-bit video pixel and a 16-bit audio sample is sufficient to encode the input range of human visions and hearings.
- Intensive I/O or memory accesses, and data locality which represent streaming nature of multimedia applications.
- Algorithms require a huge computation. In the area of video encoding/decoding, a frame rate of 30 frames per second is a normal realtime throughput requirement. A higher frame rate 60 fps or even 120 fps is common for some high-end applications.

Based on the characterization results, the past industry support for multimedia appears in three forms: application-specific processors, multimedia extensions to general-purpose processors, and multimedia co-processors. However, none of these methods can achieve both high performance and flexibility for emerging standards. Furthermore, the recent video standards like MPEG-4 and H.264 show less processing regularity and are difficult for the long existing single-instruction-multiple-data (SIMD) approach which mainly exploits explicit data parallelism in multimedia applications.

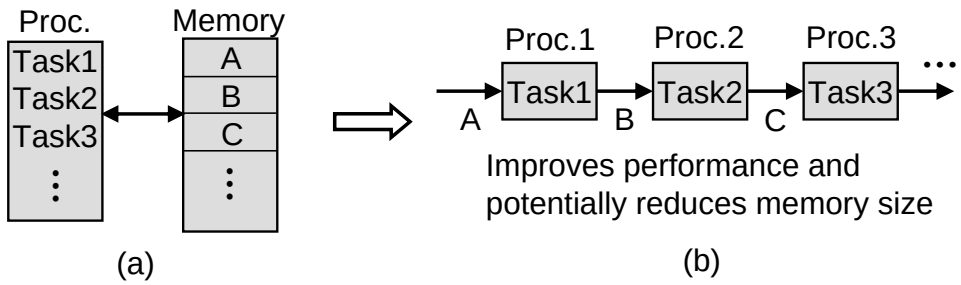



Figure 2.1: Multi-task application execution: (a) a traditional approach on a processor-memory system, and (b) a distributed processing approach using task level parallelism

### 2.1.2 Parallel Programming Model

AsAP achieves high energy-efficiency by avoiding driving global signals across a chip, centralized memories and function units, all of which are power hungry tasks. This can be accomplished by partitioning the target applications into different small tasks. Figure 2.1 shows a comparison of traditional shared memory model and a distributed processing approach using task level parallelism. This dissertation proposes a distributed processing approach to parallelize video encoding algorithms by exploiting existing locality property of video applications. The distributed processing approach can significantly reduce memory access energies because a distributed array of small local memories is far more energy efficient than a single large shared memory.

### 2.1.3 Next-generation Fine-grained Many-core System

Considering the application characteristics and combined with the power wall and memory performance gap challenges, the second project goal of this project is to design a fine-grained many-core system with the following features:

- The proposed many-core system should keep exploiting data-level parallelism with configurable SIMD style data-path. However, design trade-offs are required to keep a single core as small and efficient as possible.

- The proposed many-core system should keep exploiting task-level parallelism which is a natural property of streaming applications. Such task-level parallelization approach has advantages over SIMD approach in that it can speed up the irregular serial parts of current video standards, such as the entropy encoding in H.264/AVC.

- The proposed many-core system should achieve low power consumption. The current AsAP architecture uses globally asynchronous-local synchronous (GALS) clocking style where each processor owns their own oscillators and can stall if processors are idle [17]. The dynamic voltage and frequency scaling (DVFS) can further reduce the system power consumption [18].

- The proposed many-core system should provide a flexible low-cost topology and interconnection architecture. This project explores different low complexity topologies combined with processor shapes to further improve the throughput of current AsAP architecture.

- The proposed many-core system should offer flexible memory system for video applications. Current AsAP architecture is not efficient when algorithms require a large data memory. The proposed future system should contain a flexible configurable memory system. The design of this memory system can utilize the characterization results of H.264 video encoding algorithms.

## 2.2 Related Work

Researchers have developed various H.264/AVC ASICs for different applications ranging from mobile to high-definition television (HDTV) [19–26]. However, real-time encoding of high-definition (HD) H.264 video (up to 1080p) is a challenge to most existing programmable processors. This section surveys the related programmable approaches including: traditional DSP processors and general-purpose processors, reconfigurable com-

puting fabrics and many-core streaming processors.

### 2.2.1 Traditional DSPs and Microprocessors

Traditional digital signal processors (DSPs) have been used for media processing tasks, such as the Texas Instruments TMS320C6000 families [27]. Some other media processors are proposed specially for multimedia applications such as MPACT [28] and the Philips Trimedia architecture [29]. These media processors combine VLIW DSPs, special co-processors, video I/O and memory resources into a video processing platform. General-purpose microprocessors are also aware of the importance of multimedia application and have incorporated multimedia extensions into their architectures. All these instructions exploit sub-word parallelism available in video applications [30]. Most of them include three kinds of media instructions, data permutation and transfer instructions, SIMD ALU instructions and special instructions for specific media processing operations. Some examples of the SIMD-like multimedia extension instructions are Intel's MMX [31] and SSE [32], HP's MAX2 for the PA-RISC architecture [33], Sun Microsystem's VIS for the SPARC architecture [34], MIPS's MDMX and Motorola's ALTIVEC for the PowerPC architecture [35]. However, both DSP and microprocessors have not fully exploited the streaming nature of multimedia applications because they are designed for more general-purpose applications. They can not meet the realtime requirement in many multimedia applications and show poor energy efficiency.

### 2.2.2 Reconfigurable Computing Fabrics

A different approach is based on reconfigurable computing fabrics. Since most of the processing time in multimedia applications is spent on a small number of computation kernels, researchers have used the hardware/software co-design approach to integrate a general-purpose processor with reconfigurable co-processors or SIMD data-path to speed

up the kernel algorithms. Many research projects have explored this approach, such as RaPiD [36], MorphoSys [37] and PACT extreme processing platform [38]. The problem of this approach is that they can not scale well and the energy efficiency is still not high.

### 2.2.3 Streaming Processors and Many-core Processors

Streaming processing is proposed for applications that has - computation intensity, data parallelism and producer-consumer locality [39]. Streaming processing is firstly introduced as a programming model for chip multiprocessor (CMPs) and multi-core architectures. StreamIt [40] and Smart Memories [41] are two examples of such programming models which take advantages of the locality found in streaming applications. Based on the stream processing idea, many programmable stream processors are proposed, such as Stanford Imagine [39], RSVP [42] and SIMPil architecture [43]. These architectures normally use hierarchical structures: grouping processing elements into clusters and then those clusters are integrated into chips. A recently-fabricated streaming processor Storm-1 [44] represents the state-of-art streaming processors. Storm-1 integrates two CPU cores and a cluster of parallel integer ALUs organized into 16 data-parallel lanes with 5-ALU VLIW per lane. Strictly speaking, these streaming processors can be categorized into parallel processing architectures but not multi-core systems since they have processors acting as centralized controllers.

There exist many other many-core systems that are proposed specially for multimedia applications.

The **CELL** processor is based on a heterogeneous chip multiprocessing architecture [45]. The major goal is to improve the performance per area by reducing the size and complexity of a single core and have more cores on a single chip. The first implementation of Cell Broadband Engine (CBE) supports both scalar and SIMD execution equally well and provides a high-performance multi-threaded execution environment for all applications. CBE integrates a single 64-bit power processor element (PPE) oriented for control tasks and

eight synergetic processor units (SPEs) optimized for data and thread-level parallelism in a unified system architecture. One of the innovations in Cell is the Synergistic Processing Unit (SPU) which promotes programmability by exploiting compiler technique to target the data parallel execution primitives and also rely on statically scheduling for instruction-level parallelism. An SPU is essentially a SIMD computation engine. However, by carefully designing data alignment scheme and scalar layering, SPUs can support scalar operations very well and also can minimize the overhead of data transfer between scalar and vector operations. Instead of using caches inside the SPU, each SPU contains a local single-port SRAM unit which provides the SPU execution engine with both instructions and data. The synergetic processing drives Cell's performance. However, the programmability of heterogeneous Cell processor remains a challenging problem.

The **Ambric** processor is a homogeneous many-core processor [46]. The massively parallel processing array (MPPA) is composed of hundreds of 32-bit RISC processors and a hierarchical organization is used to combine processors into different clusters which share a large on-chip memory. Ambric MPPA model is a distributed-memory, multiple instruction, multiple data (MIMD) architecture. Ambric is similar to AsAP in the sense that both of them use communication to synchronize between different processor cores. The published running applications on Ambric include a motion estimation (ME) accelerator, a deblocking filter for real-time broadcast-quality, high-definition (HD) MPEG2 and H.264 video compression. A possible problem with Ambric architecture is that they use two synchronous registers called a channel to communicate between two processor objects. The two register buffer may not be enough for applications that have unbalanced workloads and require massive communication. AsAP also achieves higher energy efficiency than Ambric which uses synchronous clocking style without dynamic voltage and frequency scaling.

**TILE64** is a recently-proposed general-purpose 64-Core SoC Chip with mesh interconnection from Tilera Inc [47]. Each core is an identical 3-issue 32-bit VLIW DSP with 8 KB separate instruction and data cache and a unified 2-way 64 KB L2 cache. The chip uses

dynamic routers and 2D mesh topology for inter-processor communications. The first silicon chip has been reported to boot SMP linux system. TILE64 is a typical coarse-grained many-core system with dynamic network-on-chip routers.

**Fine-grained Massively Parallel Processor Based on Matrix Architecture** is proposed for mobile multimedia applications [48]. This design integrates 1 Mbit SRAM for data registers and 2048 2-bit grained processing elements connected by a flexible switching network. The target application domain in this design is image processing applications on portable devices. The proposed architecture works as a accelerator of a RISC processor in a real system. This processor have demonstrated that fine-grained many-core processors can achieve both high performance and high energy efficiency.

## 2.3 Summary

This chapter describes the goal of this research — designing and applying a fine-grained many-core system for advanced video coding. A survey of related parallel architectures for multimedia applications is presented.

# Chapter 3

# H.264/AVC Video Encoding Algorithms

This chapter gives an overview and thorough analysis of the H.264/AVC video compression standard. The amount of computation and memory requirement of underlying computation-intensive tasks have been identified and analyzed. This research suggests that video encoding composed of a transformation-based small block data-flow processing is suitable for fine-grained many-core architecture. Finally, some related parallel video encoder designs are discussed and compared with our approach.

## 3.1 Overview of H.264/AVC Video Encoding

### 3.1.1 Introduction of Video Encoding

Video encoding aims to reduce the amount of information to describe video signals. Ideally, in an lossless compression system, signals can be compressed by senders and recovered perfectly by receivers. Unfortunately, lossless approach can only achieve a modest amount of compression of image and video signals. Most practical video compression techniques are based on lossy compression, in which greater compression is achieved with the penalty that decoded signals are not identical to the original, which is tolerable by human being's vision system.

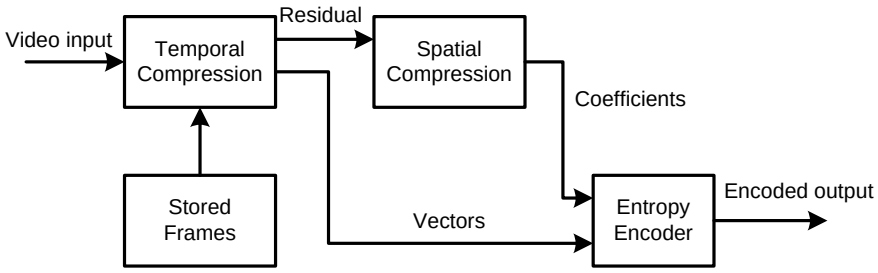



Figure 3.1: General video encoder block diagram

Figure 3.1 shows a general video encoder which has been used in most video standards including H.264/AVC standard. Basically, a video encoder consists of three major function units: a temporal compression unit, a spatial compression unit and an entropy encoder as Figure 3.1 shows [49]. The input to the temporal compression module is uncompressed video frames. The temporal compression module makes use of the similarities between neighboring frames to reduce temporal redundancy. Predicted frames are constructed by previous or future frames. The output of the temporal compression module is residual data and other coding parameters including motion vectors which are used for motion estimation. The residual frames are sent to a spatial compression unit which attempts to reduce the spatial redundancy by exploiting the similarities between neighboring samples. A transformation is applied to the residual data to convert samples into frequency domain in which they are represented by transform coefficients. The coefficients are further quantized to remove insignificant values. The output of the spatial compression block is a set of transform coefficients. The transform coefficients and the coding parameters from the temporal compression unit are sent to an entropy encoder. The entropy encoder uses variable length coding or arithmetic coding methods to remove statistical redundancy in the data and outputs a bitstream which consists of coding parameters, residual data and header information.

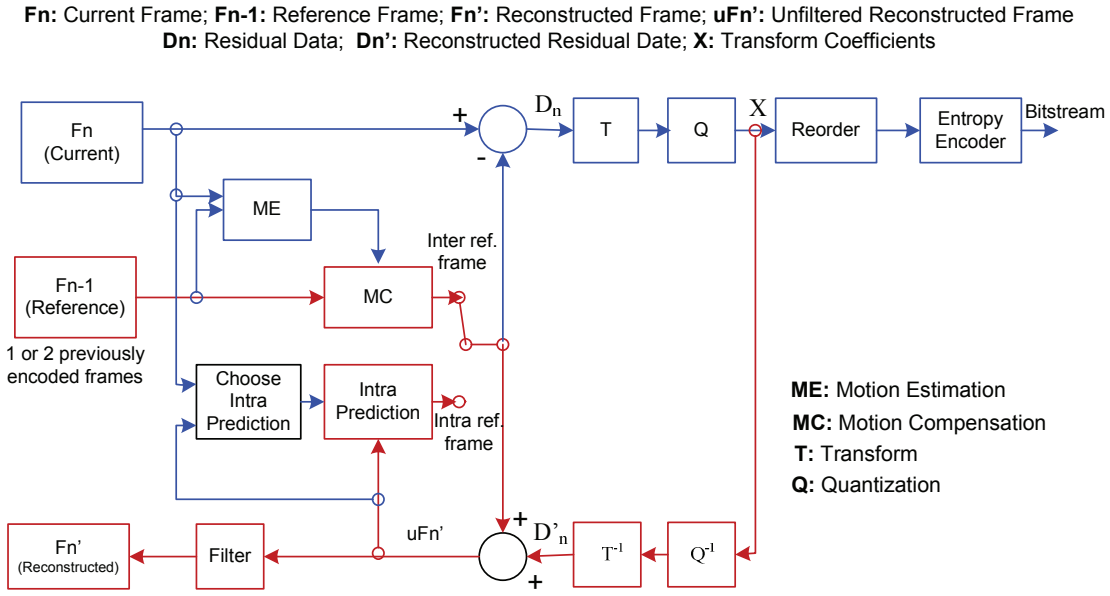



Figure 3.2: H.264/AVC encoder block diagram which includes two dataflow paths, a "forward" path (left to right, shown in blue) and a "reconstruction" path (right to left, shown in red)

### 3.1.2 H.264 Video Encoding/decoding Architecture

Image and video compression has been a very active research area for over 20 years. A lot of international image and video compression standards have been developed, including JPEG, MPEG and H.26x series standards. H.264/AVC is the latest video coding standard.

The H.264/AVC encoding follows the same video compression flow as shown in Figure 3.1. Figure 3.2 and Figure 3.3 shows the overall H.264/AVC encoding and decoding block diagram respectively. An input frame is processed in units of macro-block (a 16 x 16 block within a frame). Some of the symbols shown in the figures are:

$F_n$ denotes the current frame.

$F_{n-1}$ represents the reference frame.

$F'_n$ represents the reconstructed frame.

$P$ denotes the prediction block.

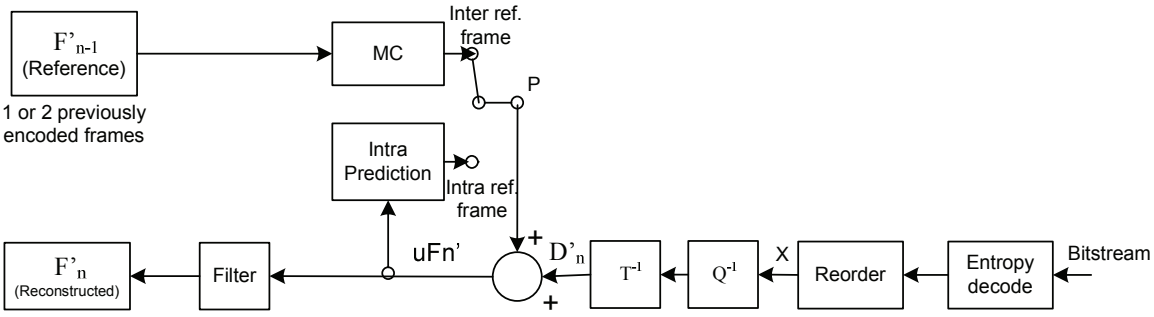



Figure 3.3: H.264/AVC decoder block diagram with similar dataflow path to the "reconstruction" path in an encoder

$D_n$ represents forward residual block.

$D'_n$ represents reconstructed residual block.

$uF'_n$ stands for reconstructed block not filtered.

$X$ stands for quantized transform coefficients.

As Figure 3.2 shows, the H.264 encoder includes two data flow path, a *Forward Path* (from left to right) and a *Reconstruction Path* (from right to left). The *Forward Path* consists of inter-frame motion estimation and compensation, intra prediction, transformation and quantization, and reorder and entropy coding. For each data block that belongs to input Frame $F_n$, the encoder first uses the previous reconstructed image samples to generate the predicted block P through intra- or inter-frame mode, and then produces the forward residual block $D_n$ by subtracting the predicted block $P$ from current block. Next, the encoder carries out the block transformation and quantization to obtain a group of quantized transform coefficients $X$. Finally, the encoder applies reorder and entropy coding operations on $X$. The entropy-coded coefficients together with side information (prediction modes, quantization parameter, motion vector information, etc) form the compressed bitstreams. The *Reconstruction* Path is made up of de-quantization, inverse transformation, and filtering module. For each input group of quantized transform coefficients $X$, the encoder first generates the reconstructed residual block $D'_n$ by de-quantization, inverse transformation

operations, and then produces reconstructed block $uF'_n$ by adding the prediction block to the reconstructed residual block $D'_n$. Next, the encoder utilizes filtering operations to reduce the effects of blocking distortion, and creates a reconstructed reference frame $F'_n$ from a series of blocks.

The H.264/AVC decoder is basically a reconstructed path of the encoder. For the compressed bit-stream, the decoder first generates a group of quantized transform coefficients $X$ by entropy decoding and reorder operation, and then uses the decoded side information to produce reconstructed residual block $D'_n$ by de-quantization and inverse transformation. Meanwhile, the decoder creates the prediction block P with the help of side information, and adds it to the residual block $D'_n$ to produce reconstructed block $uF'_n$, which is filtered to create each decoded block $F'_n$.

## 3.2 H.264/AVC Video Encoding Algorithms

H.264/AVC achieves significant video compression efficiency compared with prior standards (39%, 49% and 64% bit-rate reduction versus MPEG-4, H.263 and MPEG-2 respectively) [50]. This high coding gain increase comes mainly from a combination of new coding techniques such as inter-prediction with quarter pixel accuracy, intra-prediction, multiple reference pictures, variable block size and context-based adaptive entropy coding. The video visual quality is further increased by an in-loop de-blocking filter to reduce edge effects of block-based video coding [49]. However, all of the new techniques come with a cost of high computation complexity which makes a software approach of a real-time high-definition video encoder almost impossible in current general-purpose processors and DSPs. This section introduces the key coding blocks in H.264/AVC encoding with a focus on the complexity analysis and parallelization of the coding blocks.

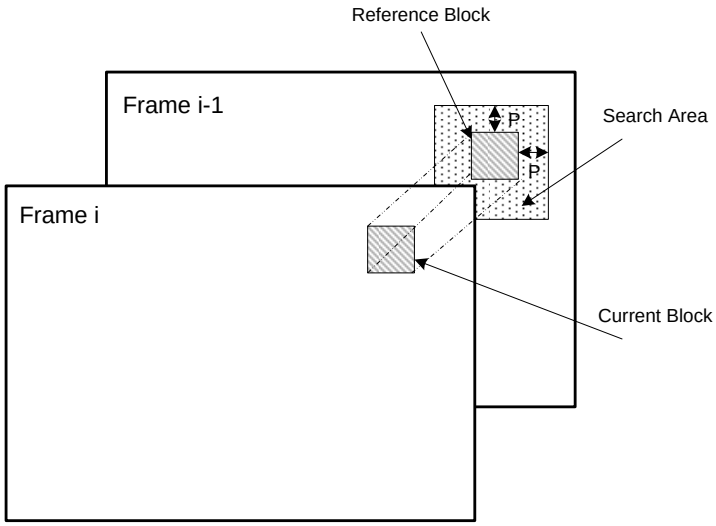



Figure 3.4: Full Search motion estimation

### 3.2.1 Inter Prediction

**Basic Idea**

Inter prediction uses block-based motion estimation and motion compensation to predict current frames based on one or more previously encoded video frames. The task of motion estimation (ME) of a macroblock is to find a 16x16-sample region in a reference frame that closely matches the current macroblock. The reference frame is a previously encoded frame from the sequence and may be before or after the current frame in display order. An area in the reference frame centered on the current macroblock position (the search area) is searched and the 16 x 16 region within the search area that minimizes a matching criterion is chosen as the "best match". The selected "best" matching region in the reference frame is subtracted from the current macroblock to produce a residual macroblock (luminance and chrominance). The residual macroblock with a motion vector describing the position of the best matching region (relative to the current macroblock position) is encoded and transmitted.

Figure 3.4 shows a full search motion estimation process. We can choose a search window where the current block is in the center of the window with a maximum horizontal

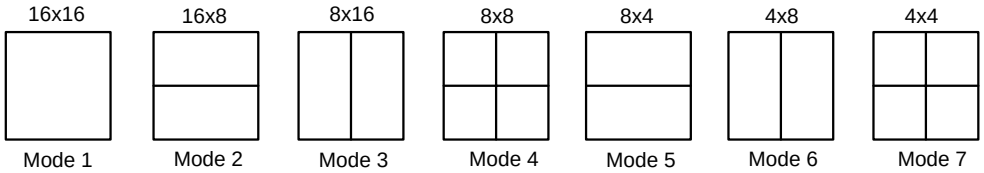



Figure 3.5: Multiple inter-prediction modes defined in H.264/AVC

and vertical displacement of $p$ pixels. A full search algorithm calculates a total of $(2p+1)^2$ cost functions, usually an SAD (Sum of Absolute Differences), to find the optimal match within the search window. If $a(x, y)$ and $b(x, y)$ are the pixels of the current and reference blocks with coordinates $x$ and $y$, and $dx$,$dy$ are the coordinates of motion vector (MV), the SAD for a $M$ x $N$-pixel block can be expressed as:

$$\text{SAD}(dx, dy) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |a(x, y) - b(x + dx, y + dy)|, (-p \leq dx, dy \leq p)$$

In H.264/AVC, the motion estimator considers multiple reference frames (up to 16 frames) and produces minimum SADs ($\text{SAD}_{min}$) and relevant MVs as the output for each 16x16 pixel block and its sub-partitions, 16x8, 8x16, 8x8, 4x8, 8x4 and 4x4 as shown in Figure 3.5. Generally, a small partition is used for video region with more details and a large partition is used for background with fewer details. The small partitions might increase the bitrate due to the necessity of coding more motion vectors. The optimal inter block size is determined based on the rate distortion (RD) costs.

In H.264/AVC motion estimation, motion vectors can be fractional numbers and this type of inter-prediction is called sub-sample prediction. Figure 3.6 shows an example of integer and sub-sample prediction. In Figure 3.6(a), a 4x4 block in the current frame can be predicted by existing samples in a reference frame (grey dots in Figure Figure 3.6(b)) if motion vectors are integers. If motion vectors are fractional values, the prediction values (grey dots in Figure 3.6(c)) are generated by interpolation between adjacent samples in the

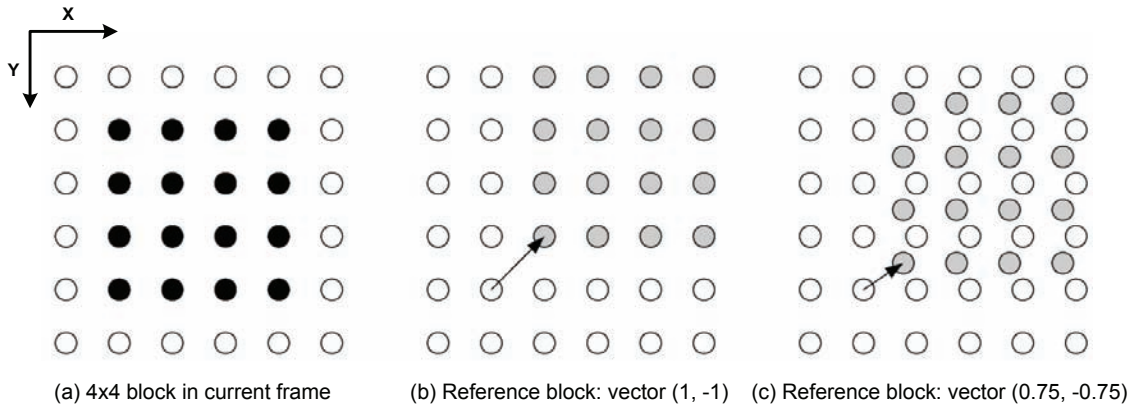



Figure 3.6: Examples of inter-prediction (a) a current 4x4 block in the current frame, (b) integer prediction where reference pixels are from existing samples in a reference frame, and (c) sub-sample prediction where reference pixels are generated by interpolation between adjacent samples in a reference frame

reference frame (white dots).

The sub-sample prediction can be further divided into 1/2, 1/4 pixel precision prediction. A higher coding efficiency in terms of Peak Signal to Noise Ratio (PNSR: unit dB) and lower bit rate is expected for a higher resolution of sub-sample prediction method.

Another complexity introduced by H.264 inter prediction is motion vector prediction. Based on the observation that motion vectors of neighboring partitions are often highly correlated, motion vectors of current blocks can be predicted by those of nearby previously coded blocks and motion vector differences (MVD). Figure 3.7 shows MV prediction based on neighboring left and top block MVs. As Figure 3.7(a) shows, E represents the current block. If the neighboring left block A, top B and top-right C blocks have the same partition size, the predicted MVs of E are medians of the MVs of A, B and C. Figure 3.7(b) shows a case where A, B and C have different block sizes. The motion vector prediction brings more dependencies between a current block and its neighboring blocks, which becomes one of the limitations for parallelization of H.264/AVC inter prediction.

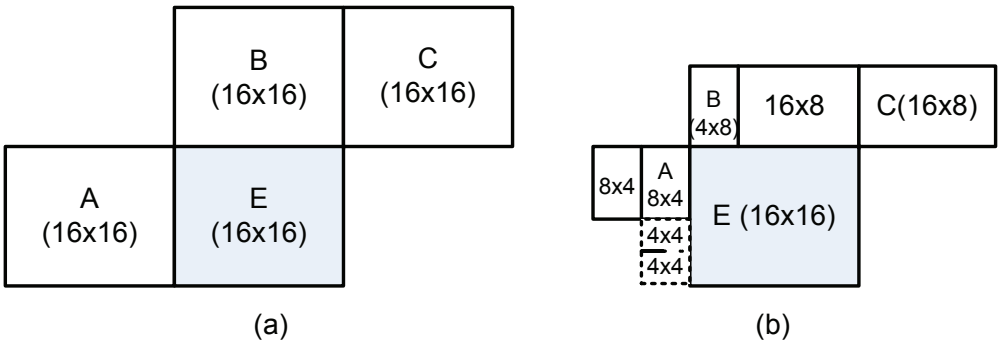


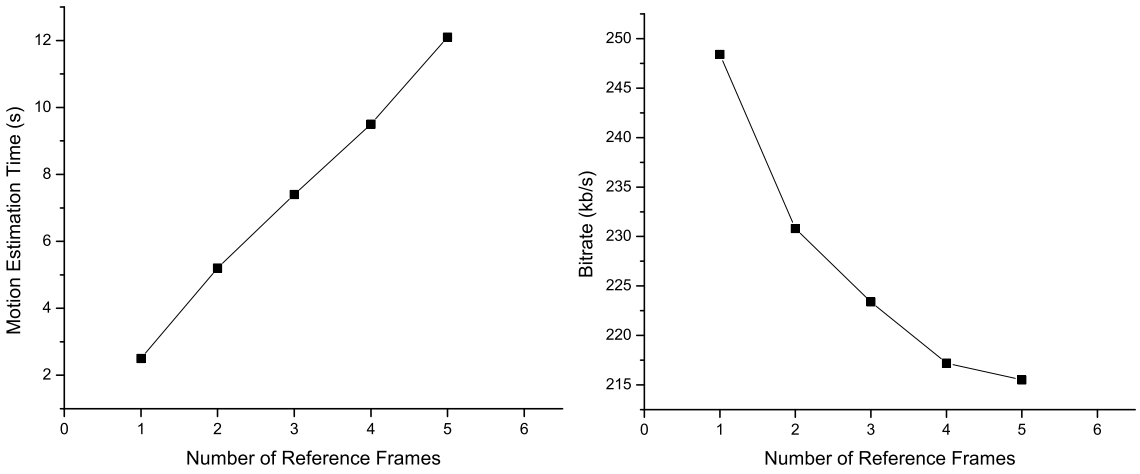

Figure 3.7: The H.264/AVC motion vector prediction for cases where current and neighboring partitions have (a) the same size, and (b) different sizes

**Complexity and Parallelization**

Intuitively, the computation time of motion estimation increases linearly if more reference frames are used or a large search range is used. As an example, implementing a full-search ME for a 30Hz CIF video (352x288 pixels), with $M = 5$ reference frames and a search range of $\pm p = 16$ pixels, requires an examination of $M \times (2p + 1)^2 = 5445$ locations for each image block and more than 16 GOPS (SAD operations). This does not take into account the computation of sub-sample prediction, motion vector prediction and variable block size determination. The motion estimation is the most computation intensive part of an H.264/AVC encoder. Many fast ME search algorithms such as three-step-search, four-step-search, 2D log-search are proposed to replace the optimal full search algorithm to reduce the computation complexity at a cost of coding efficiency (lower PNSR) [51].

In order to understand the complexity of the ME algorithms, we use the standard H.264/AVC reference software JM 12.4 [52] to conduct more quantitative experiments. A 25-frame 30 Hz QCIF ($176 \times 144$) Foreman video sequence with quantization parameter ($QP$ = 24) is used. The rate-optimization is turned off and context-adaptive variable length coding (CAVLC) is used. We have examined different motion estimation parameters and their effects in terms of processing time and bit rate.

**1) Number of reference frames:** In this experimental setup, we set the search range $\pm 16$ pixels, use all 7 block estimation modes, and conduct an optimal full search. Fig-




(a) motion estimation time (b) bitstream bitrate

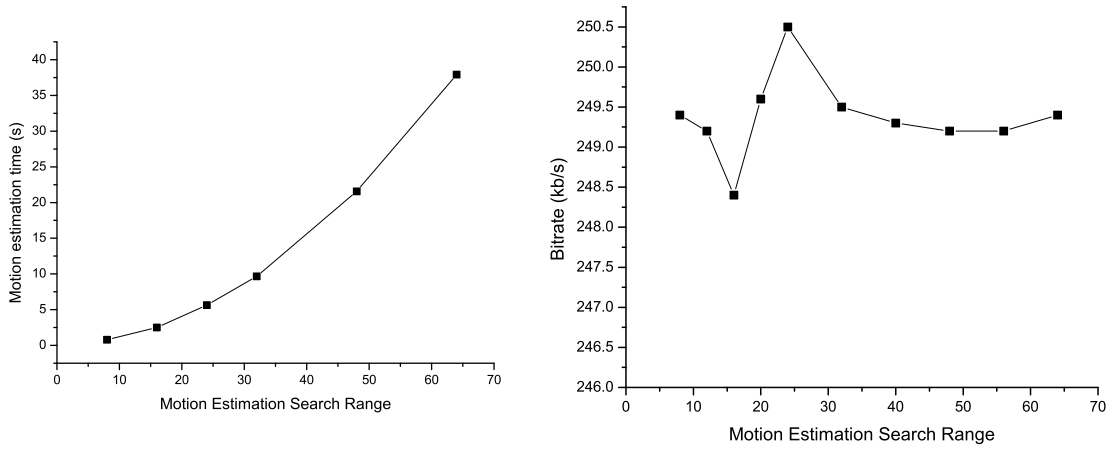
Figure 3.8: H.264 encoder performance with different number of reference pictures

ure 3.8(a) shows the motion estimation time to encode 25 frames with the number of reference frames varying from 1 to 5. If only one reference frame is used, the encoder spends 77.8% of its computation time on motion estimation. Figure 3.8(b) shows the bitrate change with different number of reference frames. Increasing the number of reference frames from 1 to 2 results in twice the computation time with only a 7% bitrate reduction. This test shows that one reference frame is sufficient for many applications.

**2) Search range:** In this experimental setup, we use one reference frame and all 7 block estimation modes as well as an optimal full search algorithm. The search range has been varied to measure encoder performance.

Figure 3.9(a) shows the motion estimation time and encoded bitstream bitrate. The ME time quadratically increases as the search range increases. Figure 3.9(b) shows an interesting fact that the optimal search range with the smallest bitrate number is not always the largest search range. In this experiment, a 16x16 search range is the optimal search range. Many adaptive search range adjustment methods have been proposed to achieve the smallest bitrate with less computation time [53].

**3) Sub-pixel sample prediction:** In this experimental setup, we use one reference frame and all the 7 block estimation modes as well as an optimal full search algorithm. The




(a) motion estimation time (b) bitstream bitrate

Figure 3.9: H.264 encoder performance with different ME search range

search range is set to be $\pm 16$ pixels. Our experiment shows that extra sub-pixel sample prediction (quarter-pixel precision) takes 27% more computation and achieves a 33.6% bitrate reduction.

**4) Variable block size:** In this experimental setup, we use only one reference frame, full search and a search range of $\pm 16$. Three different block size settings are used: 1. 16x16 block only; 2. 16x16, 8x16, 16x8, 8x8 modes only and 3. all 7 block modes. The setting 2 takes about 9.8% more computation time for a 10.6% bitrate reduction compared with setting 1. A further split of partition size in setting 3 takes 8.5% more computation time for an extra 1.3% bitrate reduction compared with setting 2. This results show that a four block size mode is enough for many applications.

**5) Full-search vs fast-search:** In this experimental setup, we use one reference frame, 16x16 search range and all 7 block modes. The fast search algorithm in JM (UMHexagon search) uses only 17.7% the computation time of a full search algorithm with a 2.7% bitrate increase. A simplified version of UMHexagon search performs even better: with around 12% computation time and a 6.9% bitrate increase. This explains why people are interested in exploring various fast search algorithms. In order to implement the inter prediction motion estimation in many-core systems, fast-search algorithms are essential to reduce

memory access and computation complexity.

Traditionally, in order to speed-up this bottleneck of video algorithms, dedicate hardware engines are used, which basically consists of (a) a parallel array of processing elements for pixel level SAD operations; (b) a local memory to exploit data reuse to reduce the external memory access; (c) an I/O control unit.

As for programmable approach, both fine-grained and coarse-grained parallelism are available in ME algorithms. The fine-grained parallelism exists within a macroblock. If both a reference frame and a current block data are loaded, the macroblock can be partitioned into different sub-blocks and each sub-blocks can be distributed into different processing elements (PEs) for parallel processing. The challenge for fine-grained parallel processing is how to distribute the inputs and collect the results from each PE and how to reuse the local memory to reduce data redistribution. Figure 3.10 shows an illustration of the fine-grained parallel mapping of motion estimation algorithm. The data of current 16x16 macroblock is distributed to 16 PEs and each PE operates on a 4x4 pixel block. The corresponding macroblock in the reference frame is also distributed to each PE. After the cost function is computed in current position, the search motion vector increments by one in both X and Y direction. In order to reuse most of the reference block data, some PEs can pass data to the PEs at bottom left, ie. PE (1,0) can pass its portion of reference frame to PE (0,1). A new row and new column data can be loaded from either on-chip or external frame buffer.

Motion estimation algorithms also have coarse-grained parallelism. At the frame level, if multiple reference frames are used, the motion estimation can operate on each reference frame in parallel. At the macroblock level, although motion estimations can operate in parallel, MVs need to be processed in raster-scan order due to the dependency introduced by motion vector prediction. Thus, MBs within the same frame can be processed concurrently only if their neighboring top and left MBs have already been encoded and reconstructed. This rule also applies to some other H.264 coding units such as intra-prediction and de-

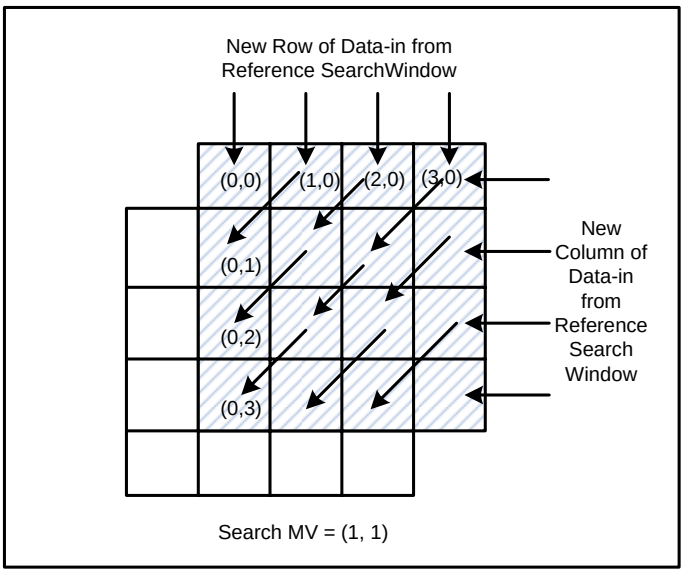



Figure 3.10: Parallel motion estimation mapping to a fine-grained many-core system

block filtering.

Overall, the regularity of the ME algorithms makes ME an ideal application for parallel processing both at fine-grained macro-block level and coarse-grained frame level.

### 3.2.2 Intra Prediction

#### Basic Idea

The intra prediction is a new feature introduced by H.264 for effective intra-block coding. The basic idea of intra prediction is to predict current block by the neighboring left and top block within the same frame. The luma intra-prediction in H.264 has two prediction strategies: intra 4x4 and intra 16x16. The intra 4x4 that predicts each 4x4 luma block individually, is well suited for images with significant details. The intra 16x16 that predicts the entire 16x16 luma block, is suitable for flat background region. The chroma uses a 8x8 prediction strategy and we call it chroma intra 8x8. There are a total of 9 modes for intra 4x4 luma block and a total of 4 modes for intra 16x16 luma block and intra 8x8 chroma block. Since the intra 16x16 luma block and intra 8x8 chroma block prediction modes are a subset of the 9 modes used by intra 4x4 luma block, we only illustrate the operations of

| M | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| I | a | b | c | d | | | | |
| J | e | f | g | h | | | | |
| K | i | j | k | l | | | | |
| L | m | n | o | p | | | | |

Figure 3.11: Labeling of prediction samples of a (4, 4) block

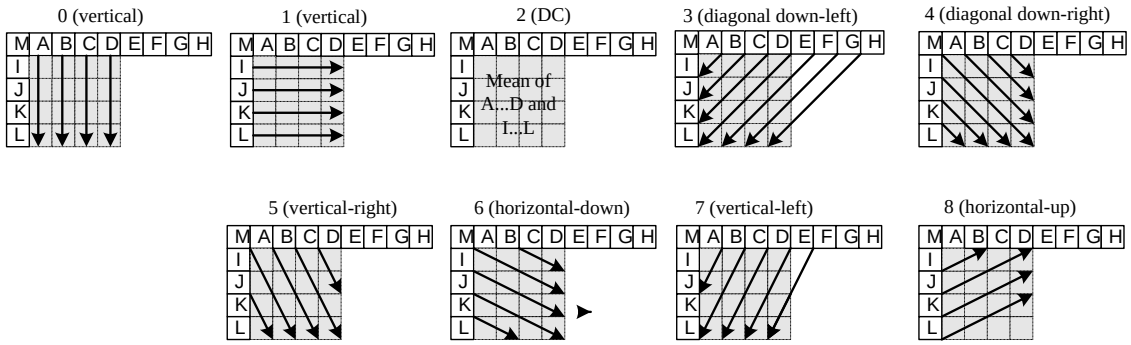



Figure 3.12: Nine 4x4 intra prediction modes

the 9 different modes used by intra 4x4 luma block. Figure 3.11 shows the labeling of the samples in a intra 4x4 block. The top 9 samples are from the three 4x4 blocks on the top of a current block. The left 4 samples are from the neighboring left 4x4 blocks.

The 9 different modes for intra 4x4 block are shown in Figure 3.12. Mode 0, 1, 2 are very straightforward which use only neighboring left or neighboring top 4 samples for prediction. For mode 3 to 8, the predicted samples are formed by using a weighted average of the samples A to M.

**Complexity and Parallelization**

The main complexity of intra-prediction arises from the calculations of all the 9 modes for intra 4x4) and 4 modes for intra 16x16 to determine the best prediction partition size and mode. Researchers have proposed many fast mode decision algorithms [54]. However, the computation of intra prediction is not as complicate as inter prediction. Besides, in a

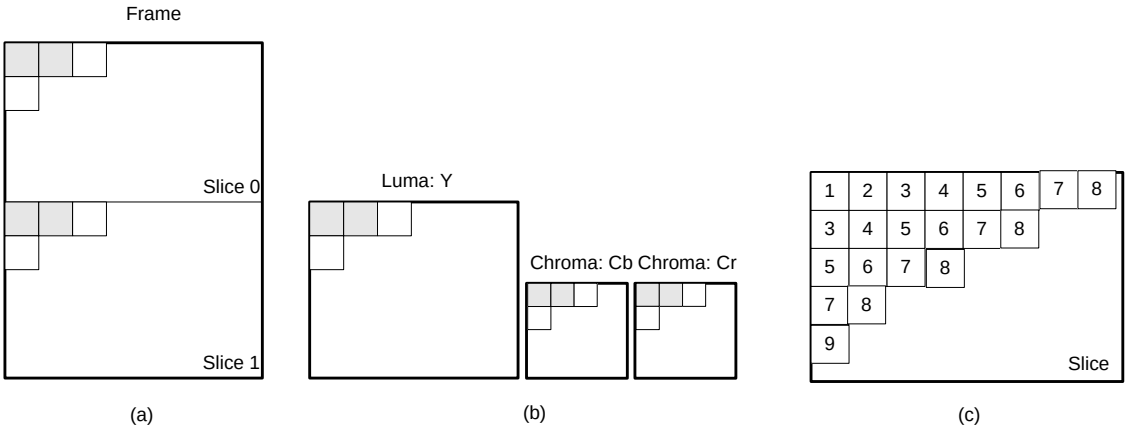



Figure 3.13: Parallelism of H.264 intra prediction (a) coarse-grained slice level parallelism (b) coarse-grained parallelism among luma and chroma encoding (c) fine-grained parallelism among each macroblock

video sequence, the number of I-frame (using only intra prediction) is usually less than the number of P or B frame (using inter prediction). We use the same experiment setup as in previous subsection. The experiment result shows that the compression ratio of inter prediction is about 5–6 times that of the intra prediction. We also examined the effectiveness of the 9 modes of the intra 4x4 prediction. We have used three different setups with: 1) 0–2 modes 2) 0–4 modes 3) all 0–8 modes. Our experiment results show that setup 2 and 3 can reduce 15% and 23% bitrate of Intra-coding frames compared with setup 1.

Three levels of parallelism can be exploited to speed up the intra prediction as Figure 3.13 shows. H.264 supports the partition of one frame into different slices which can be encoded independently as shown in Figure 3.13 (a). Unlike the inter prediction, the intra prediction of luma and chroma components can be predicted separately as Figure 3.13 (b) shows. Figure 3.13 shows a fine-grained parallelization at the macroblock level. The number of the macroblocks represents the processing order of MBs within one slice. The MBs in the same slice can be encoded at the same time. A further partition of the MBs to smaller sub-blocks is also feasible.

### 3.2.3 Transform and Quantization

#### Basic Idea

Based on the fact that H.264/AVC introduces smaller 4x4 blocks, the standard uses a 4x4 integer transform different from the 8x8 DCT (Discrete Cosine Transform (DCT)) transform adopted by previous standards. The integer transform can reduce implementation complexity and ensure drift-free property (no further noise introduced during the reconstructed path). Another feature of H.264 transform is that multiplication scaling is integrated in the quantization process [55]. A typical 4x4 block H.264 transform and quantization process can be illustrated as follows:

1) Step 1: The forward 4x4 integer transform operates on a 4x4 block $X$ and produces a 4x4 block $Y$.

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

2) Step 2: The previous 4x4 block Y is quantized individually by the following equation. $Y_{ij}$ is a coefficient of the transform described above, $Qstep$ is a quantizer step size and $Z_{ij}$ is a quantized coefficient, $PF_{ij}$ is a scaling factor from the transform stage.

$$Z_{ij} = \text{round}\left( Y_{ij} \cdot \frac{PF_{ij}}{QStep} \right)$$

In H.264, 52 $Qsteps$ are stored in a table indexed by a quantization parameter $QP$ (0 to 51). In order to avoid division operations, the above equation can be simplified as follows:

$$Z_{ij} = \text{round}\left( Y_{ij} \cdot \frac{MF}{2^{qbits}} \right)$$

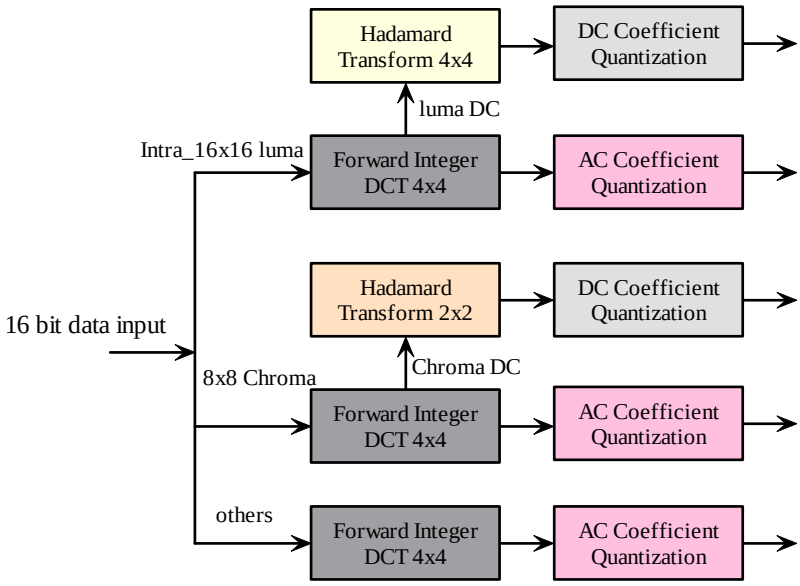



Figure 3.14: Data-flow of H.264 transformation and quantization

where

$$\frac{PF_{ij}}{QStep} = \frac{MF}{2^{qbits}}$$

and

$$qbits = 15 + \text{floor}(QP/6)$$

The above equations can be further simplified in integer arithmetic:

$$|Z_{ij}| = (|Y_{ij}| \cdot MF + f) \gg qbits$$

$$\text{sign}(Z_{ij}) = \text{sign}(Y_{ij})$$

For improved compression efficiency, H.264 also employs a hierarchical transform structure, in which the DC coefficients of neighboring 4x4 transforms are grouped in 4x4 blocks and transformed again by a second-level transform if an MB uses the intra 16x16 prediction mode. Figure 3.14 shows a complete data flow of the H.264 forward transform and quantization process, which exposes explicit task-level parallelism for our proposed fine-grained many-core system.

#### Complexity and Parallelization

The main operations of H.264 forward transform and quantization in the forward path, inverse transformation and de-quantization in the reconstructed path are simple shift, addition and table look-up operations. The DCT and related modules use 17% computation of the real-time baseline encoder [56]. The memory requirement of the 4x4 transform and the quantization table is small (only 76 16-bit word for data and look-up tables).

The regularity of the transform and quantization operations makes these coding units ideal for parallel processing. Since all the transform is applied to a 4x4 block, all the 4x4 blocks within one MB can be processed in parallel. The chroma and luma components within one MB can be processed in parallel. All the MBs in a frame slice can also be processed in parallel. In fact, the parallelism available in other tasks of the H.264 encoder limits to what extent we can parallelize the transform and quantization from a whole throughput point of view.

### 3.2.4 De-block Filter

#### Basic Ideas

De-block filtering is one of the key techniques for H.264 to achieve a high subjective quality. Since H.264 uses a 4x4 block-based integer transform and variable block size motion estimation, the de-block filter is essential to reduce the introduced blocking artifact. The de-block filter is applied to all the edges of the 4x4 blocks within one macroblock as Figure 3.15 (a) shows. The filtering is applied in an order from left to right on the vertical boundaries *a* to *d*; and from top to bottom on the horizontal boundaries in a macroblock. Figure 3.15 (b) shows samples adjacent to the boundaries of two block *p* and *q*. Each filter operation affects up to three samples on either side of the boundary depending on a parameter boundary strength ($Bs$). The filtering strength depends on the current quantifier, the coding modes of adjacent blocks and the gradient of image samples across the

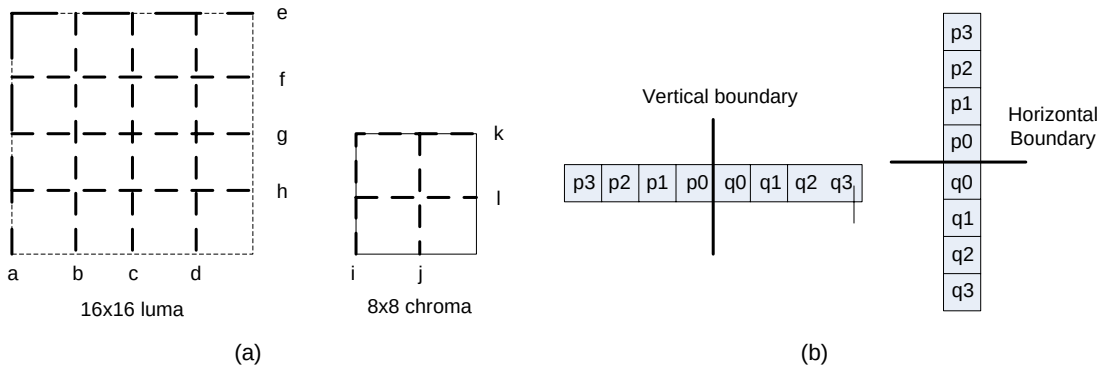



Figure 3.15: H.264 de-block filter (a) edging filter order in a macroblock (b) samples adjacent to vertical and horizontal boundaries

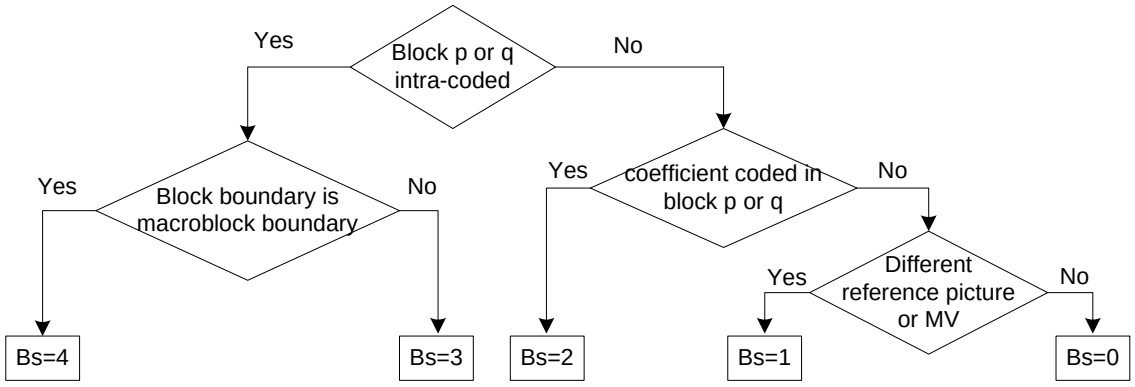



Figure 3.16: Determination of boundary strength Bs

boundary. Figure 3.16 shows the boundary strength Bs determination process. As shown in Figure 3.15, a group of samples ($p2$, $p1$, $p0$, $q0$, $q1$, $q2$) are filtered if $Bs > 0$ and $|p0 - q0| < \alpha$ and $|p1 - p0| < \beta$ and $|q1 - q0| < \beta$. $\alpha$ and $\beta$ are threshold values defined in the standard. The filters for $p0$, $p1$, $p2$ ($Bs$=4) are shown as follows.

$$p'_0 = (p_2 + 2 \times p_1 + 2 \times p_0 + 2 \times q_0 + q_1 + 4) \gg 3$$

$$p'_1 = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2$$

$$p'_2 = (2 \times p_3 + 3 \times p_2 + p_1 + p_0 + q_0 + 4) \gg 3$$

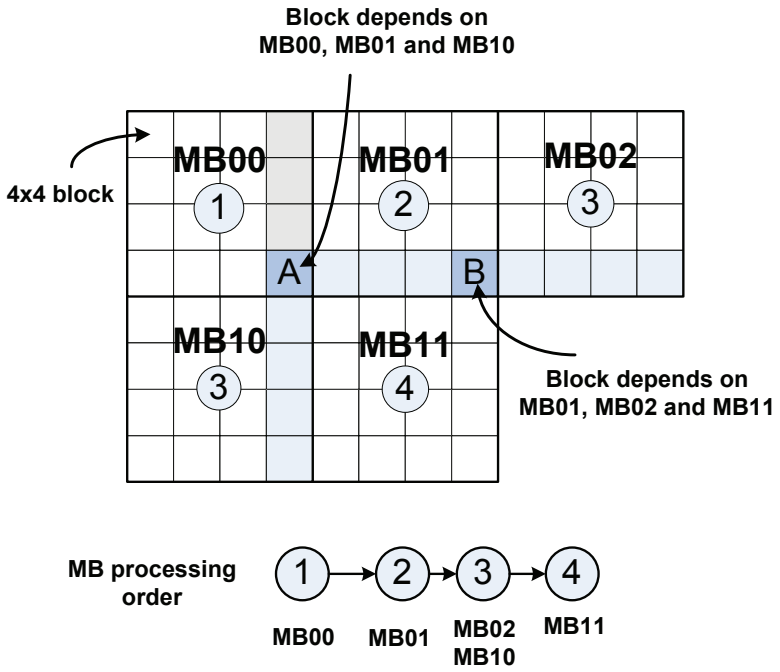



Figure 3.17: Examples of macroblock level parallelism of the H.264 de-block filtering where current macroblock depends on top and top right macroblocks

### Complexity and Parallelization

In H.264, de-blocking filter exists both in encoder and decoder and contributes to a considerable amount of computation especially at the decoder side. In a baseline encoder, around 7% of the computation time is reported for de-block filtering [56]. In the decoder, 36% of the computation goes for the deblock filtering where 40% is spent on $Bs$ calculation and 60% is spent on filtering operations [57]. The complexity of the de-block filtering mainly comes from the conditional operations in the inner loop of the algorithm and the irregular data access due to adaptive deblocking. The filter unit also requires a lot of memory access since all the reconstructed frames need go through the filter.

The challenge of parallelizing the de-block filter is how to partition the tasks into different sub-tasks so that each sub-task can operate on a small set of data. Two levels of parallel partitions are available for parallel processing. At the macroblock level, the algorithms can be partitioned in the same way as the intra prediction. As Figure 3.17 shows,

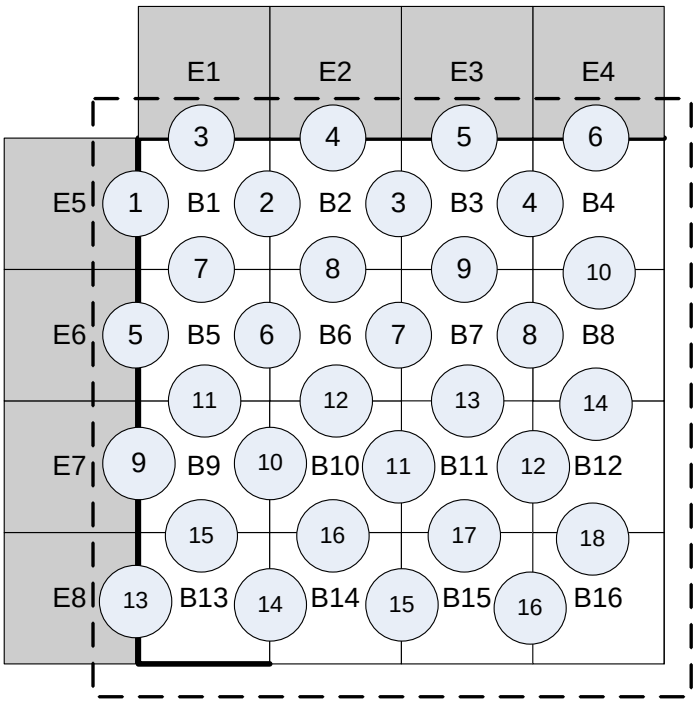



Figure 3.18: A concurrent processing order of the de-block filter within one macroblock

macroblock MB00, MB01, MB11 are at the row 0 of a frame and MB10, MB11 are at the row 1. The number below each macroblock shows a tight parallel processing order for each macroblock. Filtering MB10 may need to wait until MB00 and MB01 are filtered. However, since the de-block filtering processes the vertical boundaries first as Figure 3.15 shows. The vertical boundary filtering of MB10 can proceed concurrently with MB00 and MB01 and then waits for the finish of the MB00 horizontal filtering and the MB01 1st vertical boundary filtering. In this way, a maximum concurrency of the de-block filtering can be exploited for parallel processing.

Within a macroblock, fine-grained parallelism exists at all the edges between 4x4 sub-blocks. However, due to the fact that the edge processing has to follow the exact order as specified in Figure 3.15(a), there are data dependencies between the edge filtering. Figure 3.18 shows one of the processing order which maximizes the concurrency between the 32 edges within one macroblock. The order shows the earliest cycle that corresponding vertical and horizontal edges can be processed.

### 3.2.5 Entropy Coding

In H.264, each 4x4 block of quantized transform coefficients is mapped to an array of 16 elements in zig-zag order. The data are sent to an entropy coding unit which uses either context-adaptive variable-length coding (CAVLC) or context-adaptive binary arithmetic coding (CABAC) depending on the encoder profiles. Generally, CABAC achieves a 9%-14% bitrate reduction with a higher computation compared with CAVLC [58]. Both CAVLC and CABAC contain serial operations that are difficult to parallelize in existing programmable processors. Parallelizing CABAC is more challenging than CAVLC because of its bit-serial operations. That explains why all previous CABAC architectures are hardware-based to our best knowledge. However, we can exploit the task parallelism among CAVLC encoder for parallel processing. A more detailed parallel implementation and performance comparison of CAVLC on the fine-grained many-core system is introduced in the next chapter.

## 3.3 Related Work

Many coarse-grained parallel multi-core approaches have been proposed for H.264/AVC encoding. Most of them exploit thread-level or frame-level parallelism in video encoding algorithms. Chen et al. propose a parallel H.264/AVC encoder utilizing multi-level threading [59]. Their results show good speedups ranging from 3.74x to 4.53x over well-optimized sequential code on a quad-core system. Roitzsch proposes a slice-balancing technique for H.264 video decoding by modifying only the encoding stage and reports a performance speedup of up to 4.7 [60]. Rodriguez et al. use message passing parallelization at GOP (Group of Pictures) and frame level to speed up H.264/AVC encoding [61]. Zhao et al. present a wavefront parallelization method for H.264/AVC encoding [62]. Their parallelization method is conducted at both frame and macroblock level. Sun et al. propose a similar parallel algorithm based on a wavefront technique [63]. They partition one frame

into different macroblock regions which are processed independently. The macroblocks within the macroblock region are then parallelized with the wavefront technique.

Stream processing has been proposed for multimedia applications that have computational intensity, data parallelism and producer-consumer localities. The stream model was first proposed by Hoare in communicating sequential processes (CSP) [64]. With the rapid development of IC technology, many architectures and processors supporting stream models have emerged, such as Imagine [65] and RAW [66]. Khailany et al. use concurrency between stream commands, data parallelism, instruction-level parallelism and subword SIMD parallelism to speedup H.264/AVC motion estimation and deblocking filter kernels to achieve realtime 1080p HDTV encoding [67].

There is also a trend to use graphics processing units (GPUs) to accelerate video applications. Cheung et al. present an overview of video encoding and decoding using multicore GPUs [68]. Chen et al. implement H.264/AVC motion estimation on a GPU and report a 12 times speedup versus general-purpose CPUs [69]. However, GPUs are more suitable for applications with abundant explicit thread-level and data-level parallelism and are less efficient for some serial video encoding algorithms in the H.264/AVC standard.

# Chapter 4

# A Parallel 1080p H.264 Baseline Residual Encoder

This chapter targets energy-efficient H.264 baseline encoding from low resolution to HD video encoding on a fine-grained many-core architecture. Our programmable approach achieves both high performance (up to real-time 1080p) and flexibility. We focus on the parallelization of the H.264/AVC baseline residual encoder which utilizes integer transform, quantization and context-adaptive variable length coding (CAVLC) to encode residual data from intra and inter prediction procedures. The integer transform and quantization are well suited for parallel implementation. However, many high-performance CAVLC encoders are implemented in hardware due to its serial processing property [70, 71]. We choose to implement this software residual encoding accelerator because it is an essential task of H.264 baseline encoding. The configurable and programmable residual encoder can be used as a software co-processor for a full HD encoder.

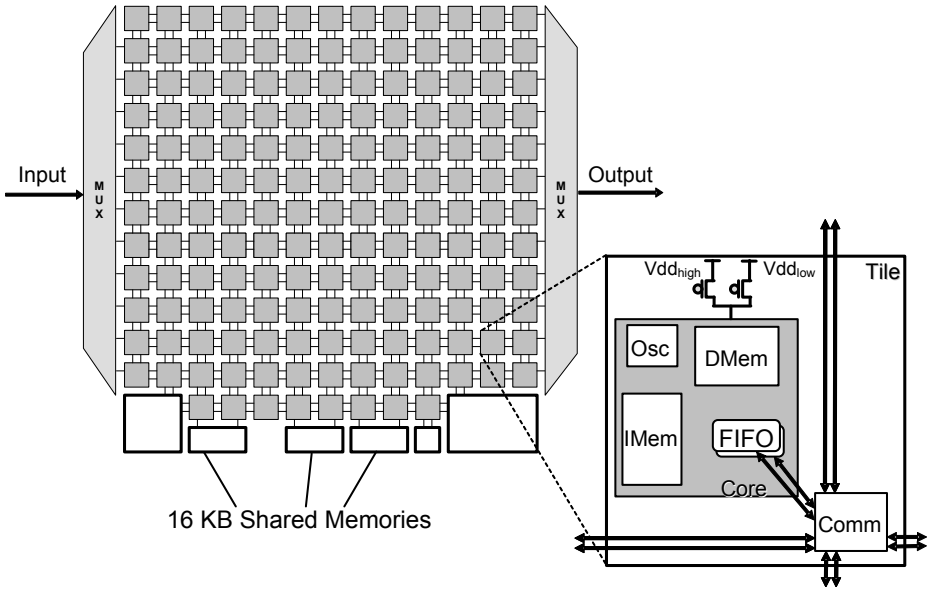


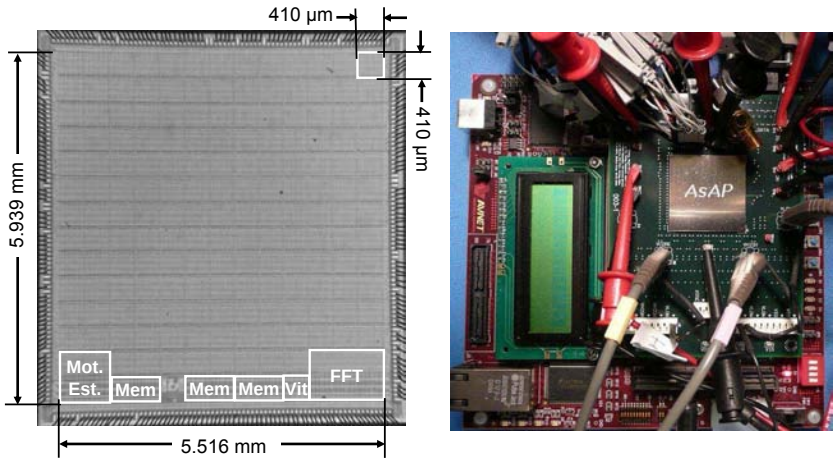

Figure 4.1: Architecture of targeted many-core system.

## 4.1 Introduction

This research demonstrates our fine-grained many-core architecture can achieve high performance and energy efficiency for both video encoding algorithms with high data-level parallelism like integer transform and quantization and serial algorithms with fine-grained task-level parallelism like CAVLC. We propose a distributed processing approach to parallelize the H.264/AVC residual encoding at 4x4 block level. The proposed fine-grained parallelization exploits the existing locality and streaming nature of H.264/AVC residual encoding algorithms. Our work differs from previous research in that we apply a fine-grained approach to exploit task-level parallelism in H.264/AVC encoding.

The fine-grained parallelization brings challenges for programmers in terms of memory, mapping, throughput and power optimizations. Our programming methodology yields an H.264/AVC residual encoder capable of realtime 1080p (1920x1080) HDTV encoding with both higher energy efficiency and area efficiency compared with other software approaches in common DSPs and customized hybrid multi-core architectures.

The rest of this chapter is organized as follows. Section 4.2 introduces the features of




(a) Die microphotograph (b) Testing board

Figure 4.2: A fully-functional AsAP chip in 65 nm CMOS which runs at a maximum of 1.2 GHz and 1.3 V.

the targeted many-core system and the corresponding parallel programming methodology. In Section 4.3, the H.264/AVC residual encoding algorithms including transform, quantization and CAVLC encoding are described and analyzed. Section 4.4 presents the approach to parallelize the residual encoding kernel in terms of partitioning, mapping and optimization. Section 4.5 shows the performance analysis and results. Section 4.6 concludes the chapter.

## 4.2 The AsAP Architecture and Programming Methodology

### 4.2.1 Many-core Array Architecture

The target AsAP (Asynchronous Array of Simple Processors) architecture is a fine-grained many-core system which is composed of simple cores that operate at independent clock frequencies and contain small memories for high energy efficiency [11].

The AsAP platform targets applications which can be partitioned into small tasks running separately on small and simple processors [72]. A second generation design allows

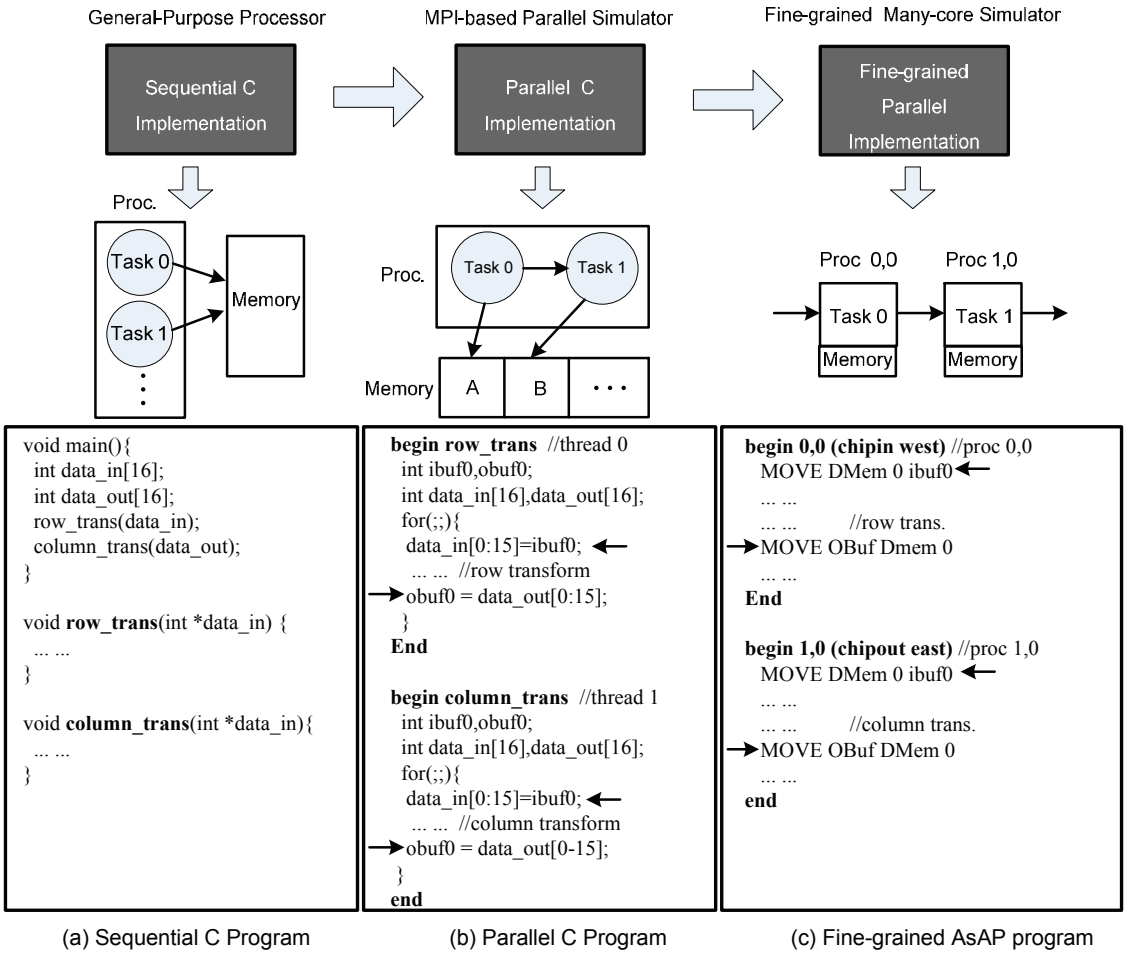



Figure 4.3: A fine-grained parallel programming methodology with corresponding multi-task application execution models and integer transform code examples.

processors to operate at independent supply voltages and contains 16 KB shared memories [73].

Figure 4.1 shows a high-level diagram of the AsAP chip which is fabricated in 65 nm CMOS technology. Figure 4.2 shows the AsAP chip die microphotograph and test board. The system is composed of 164 16-bit homogenous DSP processors, three dedicated accelerators and three 16-KB integrated shared memories, all of which have local clock oscillators and are connected by a reconfigurable globally asynchronous locally synchronous (GALS) clocking style mesh network [74]. Compared with synchronous and mesochronous on-chip communication approach [75], the GALS simplifies the clock design, provide easy scaling into future deep submicron technologies and increase energy efficiency.

Each DSP processor contains a 16-bit datapath with a 40-bit accumulator and 128 word instruction and data memories. Although processors are not tailored specially for video encoding, they handle residual encoding very well since most of the encoding tasks require very small amounts of instruction and data memories. Processor tiles are connected through configurable nearest-neighbor or long-distance links.

In our platform, each processor can run at one of two supply voltages $V_{ddHigh}$ and $V_{ddLow}$ and optimized clock frequencies. This per-core based supply voltage and frequency configuration feature is useful for achieving maximum power efficiency in video applications with dynamic workloads as demonstrated in Section 4.5.

### 4.2.2 Parallel Programming Methodology

Figure 4.3 shows the parallel programming methodology for the proposed video encoder. The methodology is divided into three steps, which is further illustrated with corresponding multi-task application execution models and examples of integer transform composed of row and column transform tasks.

We first implement a sequential C program, which uses a traditional shared memory model on a general-purpose processor as shown in Figure 4.3(a). The integer transform tasks are implemented as C functions. The algorithm is fully verified to ensure bit-level correctness compared with H.264/AVC JM software [76].

Then the sequential algorithm is partitioned into multiple parallel tasks which are implemented with simple C programs separately as shown in Figure 4.3(b). The integer transform can be divided into two tasks, row and column transforms. The two tasks can be combined by linking their inputs and outputs using a GUI-based mapping tool. We have developed a linux-based parallel simulator based on message passing interface (MPI) library to verify the parallel C implementation. All the partitions in this level are coarse-grained and have no constraints on available resources including data and instruction memory.

Then coarse-grained tasks are repartitioned to fit on the resource-constrained AsAP

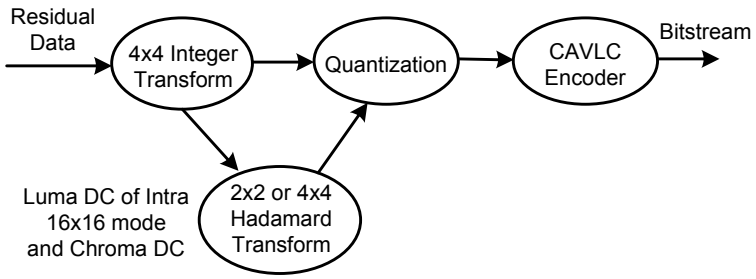



Figure 4.4: Residual data encoding procedure in an H.264/AVC encoder.

processors. As shown in Figure 4.3(c), the row and column transforms are implemented on individual AsAP processors. The final encoder is simulated on the configurable Verilog RTL model of our platform using Cadence NCVerilog. By using the activity profile of the processors reported by the simulator, we evaluate its throughput and power consumption. The distributed processing approach is suitable for video and communication applications with streaming features so that large shared memories are avoided and each processor can work on its own piece of data.

## 4.3 Residual Encoding in H.264/AVC

Figure 4.4 shows the residual data encoding procedure in the H.264/AVC baseline profile. First, a 4x4 Integer Transform (IT) is applied to the residual data from either intra or inter prediction procedures. For the intra 16x16 prediction mode, an additional 4x4 Hadamard Transforms (HT) is applied to the 16 luma DC values within one macroblock. If the residual data are chroma DC coefficients, a 2x2 HT is applied. The CAVLC encoder encodes the zig-zagged 4x4 or 2x2 quantized transform coefficients and sends the bitstream out. The integer transform and quantization has been described in previous chapter and Figure 4.4 only describes the CAVLC encoder in detail.

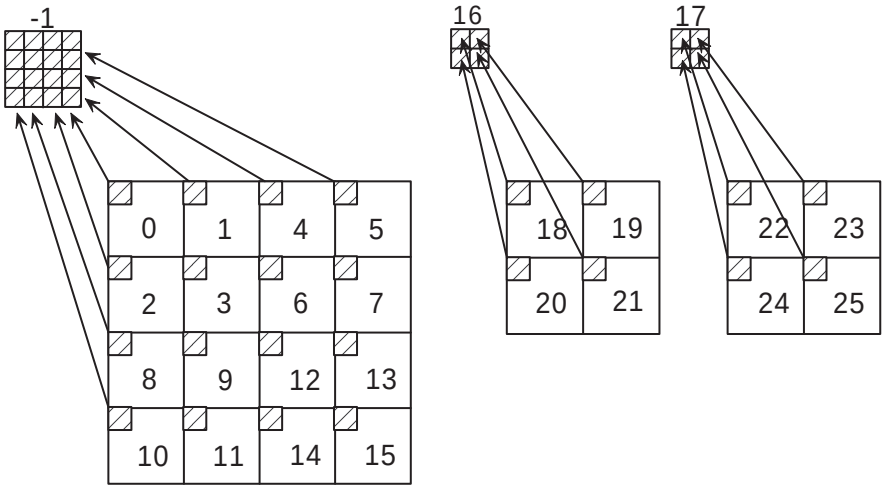



Figure 4.5: Scanning order of residual blocks within a macroblock.

Table 4.1: Elements of CAVLC Encoding per Block

| **Elements** | **Description** |
|---|---|
| *Coeff_token* | Encodes the number of nonzero coefficient and number of signed trailing ones - one per block |
| *Sign_trail* | Encodes the sign of trailing ones one per trailing ones maximum 3 per block |
| *Levels* | Encodes the remaining nonzero coefficients one per level excluding trailing ones |
| *Total_zeros* | Encodes the total number of zeros before the last coefficient - one per block |
| *Run_before* | Encodes the number of run zeros preceding each nonzero levels in reverse zigzag order |

### 4.3.1 CAVLC Encoding

The CAVLC encoder is used for encoding transformed and quantized residual coefficients of one video macroblock in the processing order as shown in Figure 4.5. A maximum of 27 blocks must be encoded within one macroblock. Block "–1"contains 16 Luma DC coefficients if the current macroblock is encoded in 16x16 intra mode. Blocks 16 and 17 are formed by the DC coefficients of two Chroma components.

The CAVLC encoder can be partitioned into scanning and encoding phases. In the scanning phase all of the blocks are scanned in zigzag order. In the encoding phase, five different types of statistic symbols are encoded sequentially using look-up tables as Table 4.1 shows.

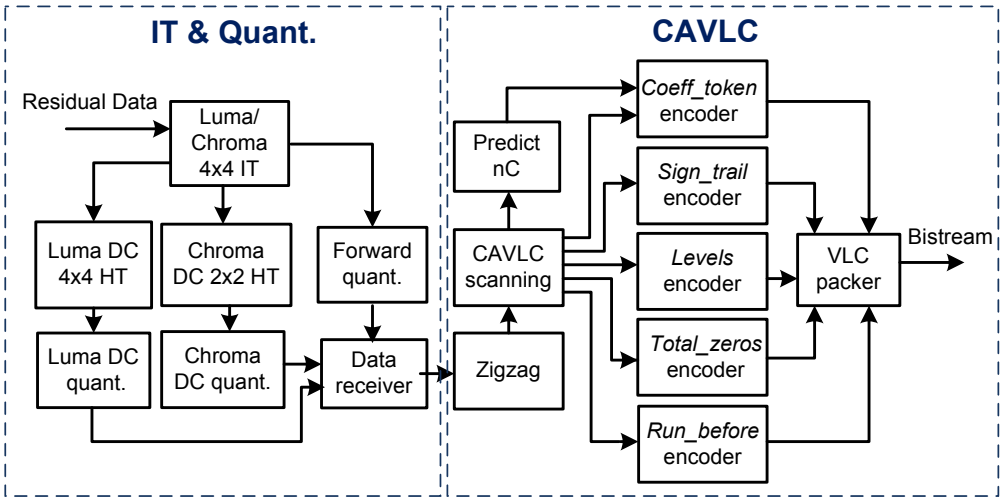
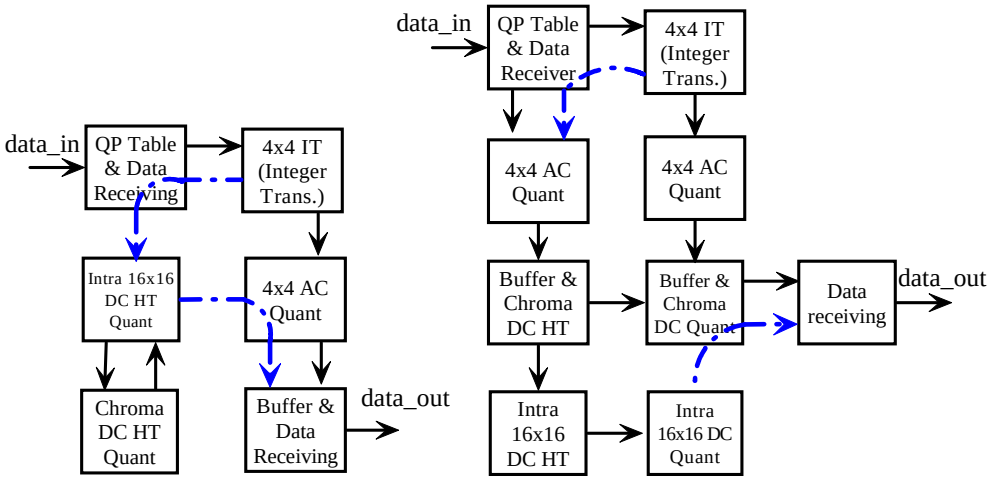



Figure 4.6: Data flow diagram of the proposed H.264/AVC residual encoder.

The complexity of CAVLC mainly comes from the context-adaptive encoding of the first and third elements, *coeff_token* and *levels*. The *coeff_token* is encoded for the total number of nonzero coefficients and trailing ones. Five different VLC tables are available for *coeff_token* encoding and the choice of table depends on the number of nonzero coefficients in the neighboring left and top blocks. This data dependency requires a large memory to store the number of nonzero coefficients for high quality video encoding. The *levels* are the nonzero coefficients (excluding trailing ones) encoded in reverse zigzag order. The *levels* code is made up of an all 0 prefix followed by a symbol 1 and suffix. The length of the suffix is initialized to 0 unless there are more than 10 nonzero coefficients and less than 3 trailing ones, in which case it is initialized to 1. The length of the suffix can be adaptively incremented if the current level magnitude is larger than a certain threshold. A maximum of 6 bits are used for suffix encoding [77].

## 4.4 The Proposed Parallel Residual Encoder

Figure 4.6 shows the data flow of the proposed parallel residual encoding kernel. The input residual data are sent to the shared 4x4 integer transform module. Then the transform coefficients are forwarded to the AC quantization, Chroma DC and Luma Intra 16x16




(a) Non-optimized

(b) Optimized with almost two times higher throughput

Figure 4.7: Two mappings of integer transform and quantization.

Hadamard Transform and quantization modules separately. All the quantized coefficients are collected by the data receiver module and sent to the CAVLC encoder. In the CAVLC encoder, the zigzag and CAVLC scanning block are the first phase of processing. Then corresponding syntaxes are distributed to five different encoding units in parallel. The packing unit collects and packs the final codes into an output bitstream. When implementing the encoder on the array processor, each task is first mapped to a single processor to allow parallel execution. If either more memory or high performance is required than can be provided by a single processor, the task is mapped to multiple processors. Code for each processor is implemented independently, considering only its inputs and outputs. Once the mapping and communication patterns are determined, coding for the small-memory processing array is similar to writing codes for a sequential machine. However, an efficient parallel mapping of this application on a fine-grained architecture still requires overcoming some challenges in terms of memory usage, mapping and throughput optimization. The following subsection describes our approach to these problems.

### 4.4.1 Integer Transform and Quantization

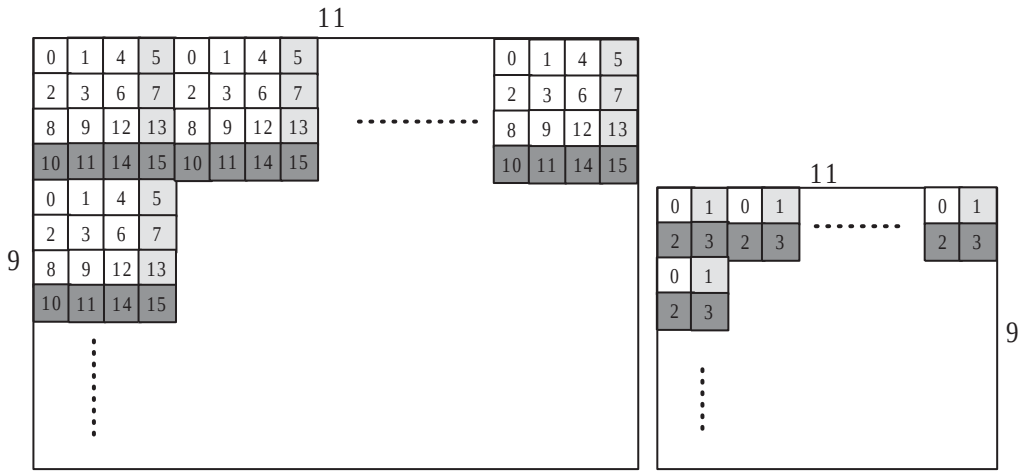
#### Memory and Algorithm Optimization

Since the proposed encoder works at the 4x4 block level, most of the time a 16-word memory is required for storing streaming data. Thus, the 4x4 integer transform can be directly implemented on one AsAP processor in 97 cycles to process each 4x4 block (without configuration overhead). As for the quantization, we use look-up tables to implement computations such as $QP/6$, $QP$ mod 6, $2^{qbits}/6$ and $2^{qbits}/3$. Another problem for quantization is that the size of intermediate values exceeds 16 bits due to the large size of multiplication factors. This can be solved by using the 40-bit accumulator to store the intermediate values so that a maximal precision is preserved during the quantization procedure.

If the macroblock is in intra 16x16 prediction mode, the luma DC (block –1) are first sent to the CAVLC encoder as shown in Figure 4.5. This processing order breaks the natural task-level pipeline because the DC values can not be fully collected until all the luma AC blocks within one macroblock are transformed and quantized. Thus, we need to buffer a maximum of 256 quantized luma AC values to reorganize the block order. We can compress two 8-bit AC values into one 16-bit word so that the buffer tasks can be implemented on one processor. Similarly, a maximum 64 quantized chroma AC values must be buffered so that the Chroma DC values can be sent first (block 16 and 17 in Figure 4.5).

#### Mapping and throughput optimization

Figure 4.7(a) shows a 6-processor direct mapping of the integer transform and quantization procedures on the array processor. The two dashed lines represent long-distance links. The chroma DC HT and quantization procedures are implemented in a single processor. This fine-grained mapping creates an application level pipeline so that the major transform and quantization tasks are running in parallel. Our initial evaluation shows quantization is the bottleneck of this mapping. In order to support HDTV 1080p at 30 fps,




(a) Luma (b) Chroma Cb or Cr

Figure 4.8: Macroblocks in a QCIF frame.

the 4x4 AC quantization processor needs to operate at 2.14 GHz. Figure 4.7(b) shows a 9-processor mapping. We have duplicated the *4x4 AC Quant* unit to double the throughput of the quantization tasks. The transformed coefficients are sent from the *4x4 IT* processor alternately to the two *4x4 AC Quant* processors. The chroma DC HT and quantization are implemented in two processors which also buffer half of the Luma and Chroma AC blocks within one MB. The intra 16x16 DC HT and quantization are running on two processors independently. The 9-processor mapping doubles the throughput with three extra processors and simple code duplication and re-mapping.

We can further parallelize the integer transform and quantization due to the vast data parallelism available in the transform and quantization operations. The residual encoder is parallelized in a way similar to a software pipeline. Therefore the throughput of the encoder depends on the slowest task. Since the integer transform and quantization are fast enough for 1080p video encoding, we do not need to further improve this part. In the following subsection, we focus on the parallelization of the CAVLC encoder which may be slower than the integer transform and quantization tasks in the case that a test video sequence contains many non-zero residual data.

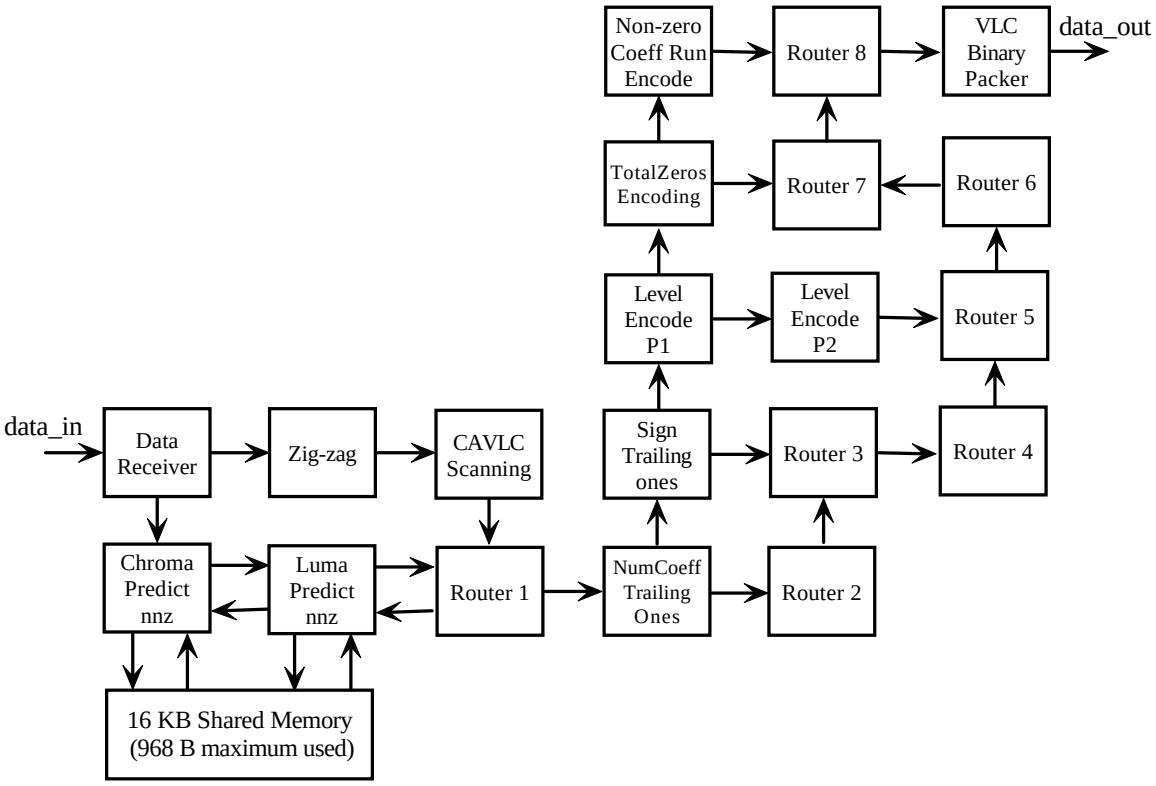



Figure 4.9: A 20-processor CAVLC straightforward mapping done manually without long-distance interconnection.

### 4.4.2 The CAVLC Encoder

Compared with integer transform and quantization, the CAVLC algorithm is intrinsically serial due to the dependencies among 4x4 blocks within one macroblock and the neighboring macroblocks within a single video frame. However, task level parallelism can still be exploited by distributing different tasks among processors [78].

#### Memory Optimization

In the CAVLC encoder, the *coeff_token* symbol (refer to Table 4.1) is encoded with a table look-up based on the number of nonzero coefficients (TotalCoeff) and trailing $\pm 1$ values (TrailingOnes). In H.264/AVC, five different look-up tables are used for this purpose and the choice of table depends on a parameter *nC* which is the average of the number of nonzero coefficients of the neighboring left and upper blocks named *nA* and *nB* respec-

tively. Figure 4.8 shows the organization of macroblocks within one QCIF frame. The gray and dark gray blocks are data-dependent blocks between neighboring macroblocks. As macroblocks are processed in raster scan order, a large memory is needed to store the number of the nonzero coefficients of those data-dependent blocks. However, as each macroblock needs only *nA* and *nB* from neighboring 4x4 blocks, the memory requirement can be reduced by maintaining a global memory of *upper_nA* and *left_nB* in one of the 16-KB on-chip memories of AsAP array. For one 1080p HDTV frame, the *upper_nA* contains 960 parameters and *left_nB* contains 8 parameters. As each parameter uses no more than 5 bits, the *upper_nA* and *left_nB* can be further compressed to save half of the memory.

In our proposed CAVLC encoder, an arithmetic table elimination (ATE) technique is used to encode level information. The level encoding starts from the last nonzero coefficient (excludes trailing ones). Two parameters, *levels* and *vlcnum*, are sent to the encoding unit in each iteration. *Vlcnum* is initialized to 0 or 1 and can be updated for the next level encoding depending on the current level magnitude. The encoding unit encodes VLC0 and VLC1–6 separately with simple shift and addition operations. Due to the limit of the instruction memory, *level* encoding has been implemented on two processors as shown in Figure 4.9. The P1 processor receives *level* information, sends *level* and *vlcnum* to P2 and updates the *vlcnum* each time.

We use look-up tables to encode the other symbols: *coeff_token*, *total_zeros* and *run_before*. Most of the data in the VLC tables are less than 4 bits except for some entries in the *coeff_token* when the number of total nonzero coefficients is larger than 12. Moreover, the VLC table used to encode *total_zeros* has a triangular structure, where most data are zeros. Based on the above observations, we can divide the tables into smaller compressed tables and then determine which table to use at run-time with little extra computation. Our approach achieves a compression ratio of 4 so that the data tables of the tasks *coeff_token*, *total_zeros* and *run_before* fit into one processor's 128-word 16-bit data memory.

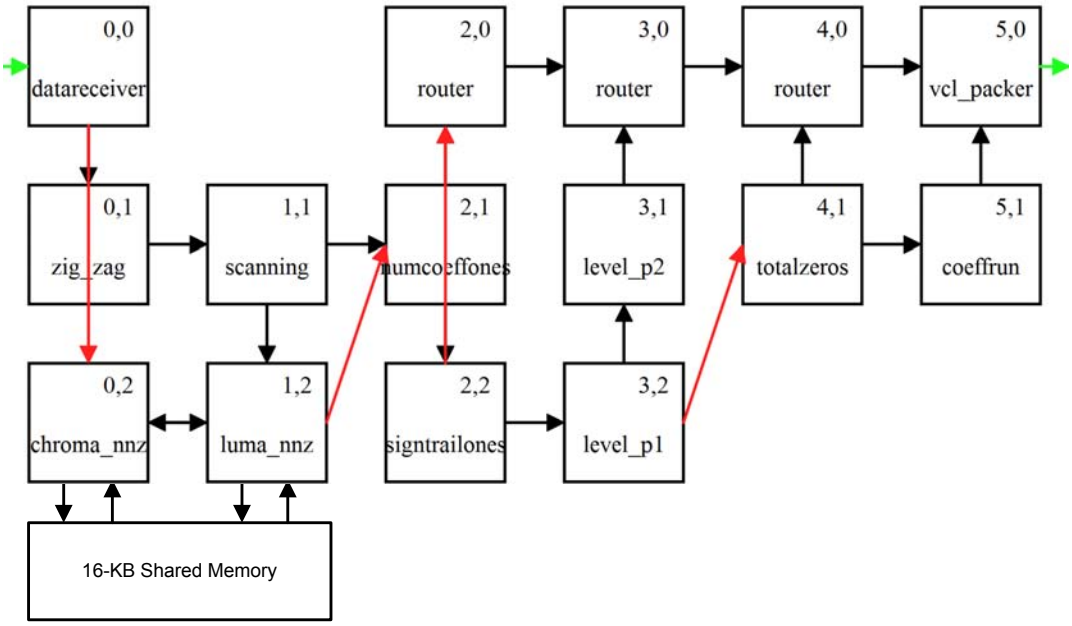



Figure 4.10: A 15-processor CAVLC mapping performed by an automatic task mapping tool [79].

**Dataflow Mapping**

As Figure 4.6 shows, the CAVLC encoder can be easily partitioned into a number of independent serial and parallel tasks. When implementing the encoder on an array processor, each task is firstly mapped to a single processor to allow parallel execution. Each processor stores only a small amount of data (up to a 4x4 block data) for local computation. It is worth mentioning that the fine-grained partition step determines the throughput of the encoder since all of the tasks are implemented in a software pipeline style. In the following step, we need to map the fine-grained task graphs into the 2D mesh array architecture. This mapping step can either be conducted manually or automatically by a customized AsAP mapping tool which aims to maximize nearest neighbor communication and insert as few number of routing processors as possible [79].

Figure 4.9 shows a 20-processor straightforward manual mapping using only nearest-neighbor connections. The CAVLC scanning unit sends statistical information only to the *coeff_token* encoding unit and the *coeff_token* encoding unit passes the information imme-

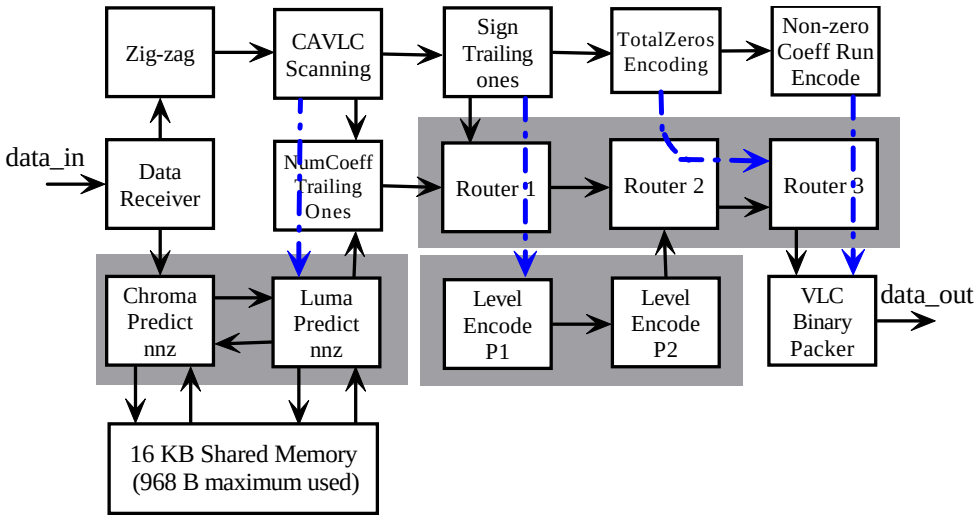



Figure 4.11: A 15-processor CAVLC mapping done manually with throughput identical to the mapping shown in Figure 4.10.

diately to the next *sign_trail* encoding unit. This takes place for every encoding unit before it begins to operate on its own portion of data. This approach simplifies the mapping and will not degrade the throughput since the code produced by each unit needs to be collected in sequential order by the VLC packing unit anyway. In Figure 4.9, the *nC* prediction unit is implemented on two processors for Luma and Chroma separately. The 16 KB shared memory supports two independent interfaces, which is ideal for this case.

The mapping in Figure 4.9 is inefficient due to the constraints of a maximum of two input ports per processor and only nearest-neighbor processor communication. Eight routing processors are required to pass data around the graph. Figure 4.10 shows a compact 15-processor automatic mapping by the AsAP mapping tool which aims to map an algorithm with the shortest interconnection links and number of routing processors. The four long arrow lines represent long-distance links. The length of all the links are less than one processor. A saving of five routing processors shows the efficiency of the low overhead long-distance interconnection architecture [80]. With a little more manual optimization, we have another similar 15-processor mapping shown in Figure 4.11, which is more regular and uses exactly a 5 by 3 processor array plus the shared memory. As shown in the shadow

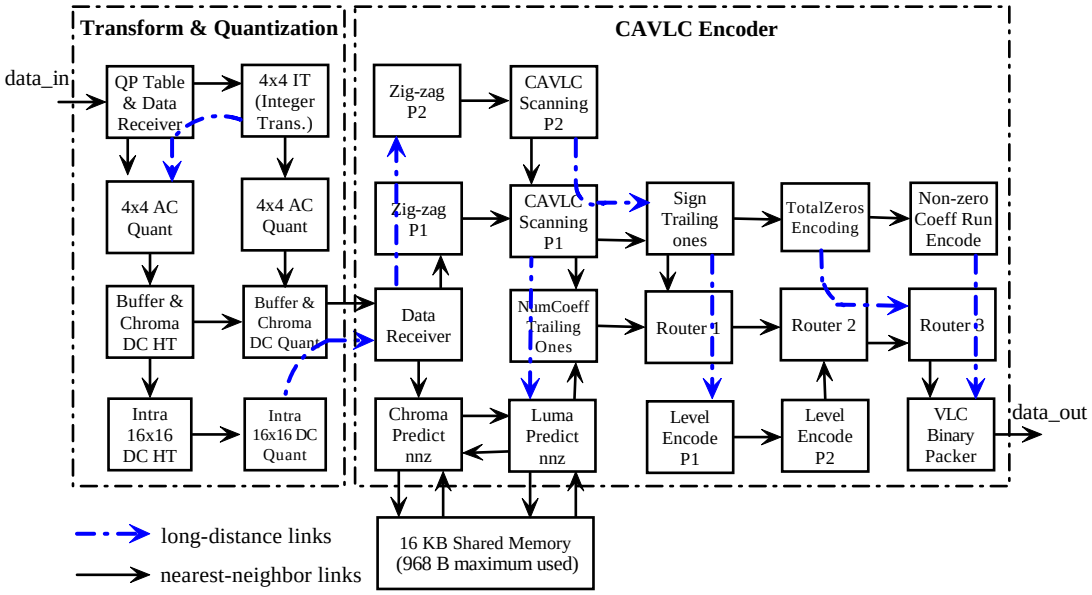



Figure 4.12: The proposed 25-processor H.264/AVC residual encoder mapping.

box of Figure 4.11, compared to the CAVLC data-flow in Fig 4.6, we added two more processors for the *nnz prediction* and *level encoding* and three routing processors which are required because of the constraints of two input ports per processor. Overall, the parallel mapping is very straightforward but effective once the algorithm are partitioned well.

### Throughput Optimization

The throughput of the 15-processor mapping can be further optimized by characterizing the workload of each processor and speeding up the processors in the critical data path. The non-critical processors add only latency to the system and do not affect the overall throughput. Since the processors stop once they finish their jobs, the processing time of one 4x4 block approximates the processor active time during the encoding.

Our evaluation shows the critical path of the CAVLC encoding includes zigzag reorder, CAVLC scanning, level encoding P1&P2, and VLC binary packing. Three methods are adopted to optimize the mapping. First, the coding of these critical path processors are optimized by using AsAP's instructions and features such as block repeat, automatic address generation and data forwarding. Second, the workload of VLC packing is re-mapped onto

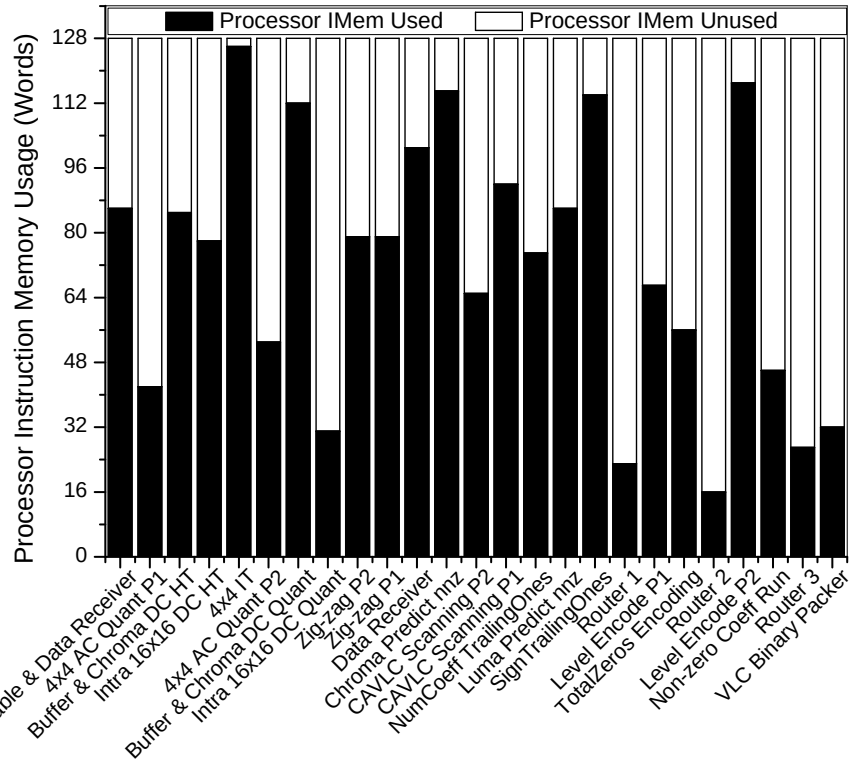



Figure 4.13: Instruction memory usage of the proposed 25-processor encoder.

routing processors. The codes can be packed as soon as they are produced by each encoding unit. Third, we add another two processors to further parallelize zig-zag and CAVLC scanning procedures as shown by the CAVLC encoder in Figure 4.12. These three optimizations triple the average throughput of the CAVLC encoder which can encode 1080p (1920x1080) HDTV at 30fps or higher for various video test sequences.

## 4.5 Simulation Results and Comparison

### 4.5.1 Implementation Results

Figure 4.12 shows our proposed 25-processor fine-grained mapping for the H.264/AVC residual encoder. A total of 8 processors are used for transform and quantization and 17 processors including one 16-KB shared memory (968 bytes maximum used for 1080p HDTV) are used for CAVLC encoding. There are eight long-distance links with a length

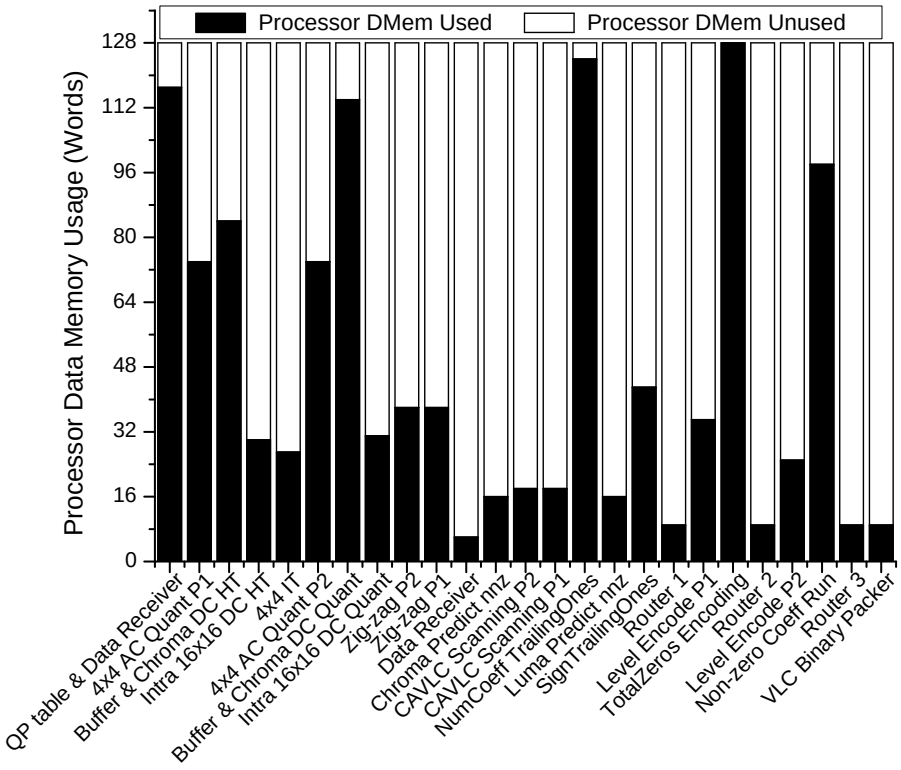



Figure 4.14: Data memory usage of the proposed 25-processor encoder.

of one processor. All other processors not included in the application mapping within the AsAP array (Figure 4.1) are turned off to save power by halting their oscillators and disconnected them from the power grid with their individual power transistors. We may use the large number of unused processors to implement other workloads such as wireless communication or encryption for some applications such as a wireless security video encoding system.

Figure 4.13 and Figure 4.14 summarize the instruction and data memory usages for each processor among the 25 processors, respectively. Our implementation shows that 128-words of instruction and 128-words of data memory are more than enough for the H.264/AVC residual video encoding. Each processor of the 25-processor encoder uses an average of 72 words of instruction memory, which is 56.3% of all available instruction memory; and an average of 48 words of data memory, which is 37.2% of all available data memory.

In our proposed residual encoder, the throughput of the transform and quantization takes a maximum of 3960 cycles to encode one macroblock. The throughput of the CAVLC encoder is highly dependent on specific test video sequences and encoding QP value. In H.264/AVC, the coded block patterns (CBP) are used to determine the all-zero residual blocks which are not necessary to be encoded. Considering the CBP effects, we performed the simulations of our residual encoder using 8 test sequences with different frame size including QCIF foreman, CIF football, 4CIF soccer, 720p stockholm, 720p shields, 1080p rush hour, 1080p pedestrian area and 1080p blue sky. All of these test sequences are encoded with four different QP values from 25 to 36.

We use the JM 12.4 reference software to encode original video sequences with a baseline setting. We collect the intermediate residual data after the intra and inter prediction in reference software and send them to our residual encoder as testing inputs. Simulation results are calculated by averaging the cycles of encoding one macroblock of one I type and one P type frame with a QP value from 25 to 36. If all the processors run at a maximum of 1.2 GHz with a supply voltage of 1.3 V, the encoder needs to encode one macroblock with less than 4902 cycles to support 1080p HDTV encoding. Figure 4.15 shows the average cycles to encode one macroblock for all the tests. As shown in Figure 4.15, all of the tests use less than 4902 cycles to encode one macroblock. The QCIF foreman test sequences has the highest computation complexity and requires 4841 cycles to encode one macroblock at $QP = 25$. All of the other test sequences have a very steady encoding throughput in terms of average cycles per macroblock within a range of 3500 to 4200 cycles per macroblock. The results indicate that the encoder meets the real time requirement of 1080p HDTV encoding at 30 fps.

### 4.5.2 Performance Evaluation

A more detailed analysis of processor execution reveals some interesting insights into the bottleneck of our design. Figure 4.16 illustrates average processor activity of the en-

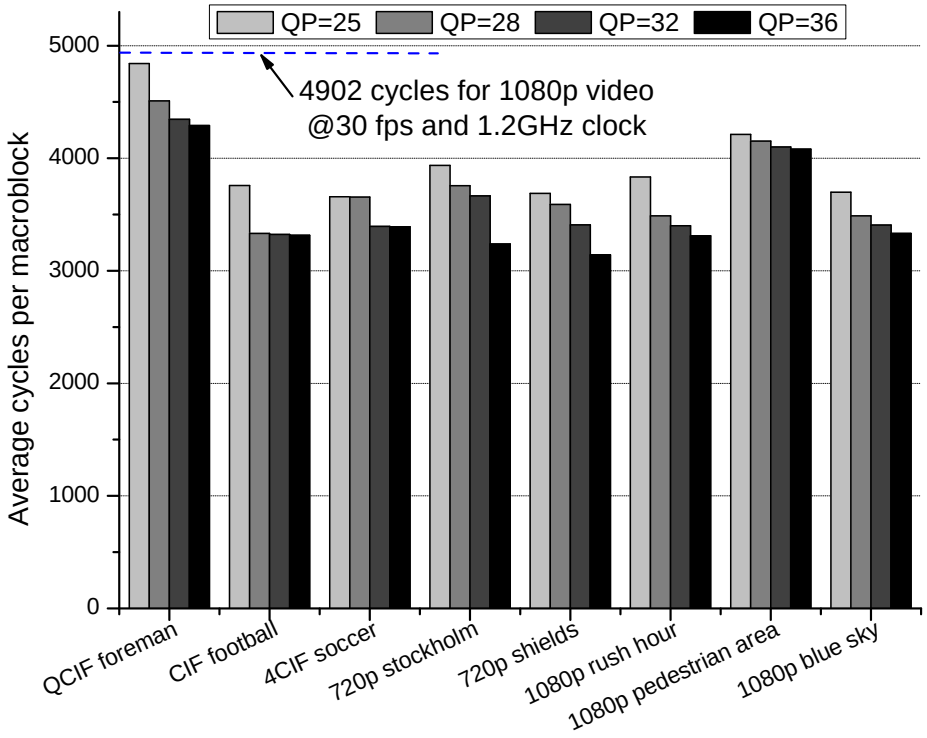



Figure 4.15: The average cycles to encode one macroblock for test sequences with varying frame sizes and QP values.

coder for encoding foreman testing video with $QP = 25$. The activity of each processor (the amount of time spent executing, instead of stalling), is indicated by the black bar in the figure. The white bar indicates the time stalled on output, while the gray bars indicate the time spent waiting for input to arrive. Figure 4.16 shows that the two *4x4 AC Quant* processors are running all the time and they are both bottlenecks of our design in this case. The two processors *Intra 16x16 DC HT* and *Intra 16x16 DC Quant* are stalling on input for most of the time because the video frames are not encoded in intra 16x16 mode. The *QP Table & Data Receiver* and *4x4 IT* processors stall on output for more than 30% of the whole encoding time because the downstream *4x4 AC quant* processor is not fast enough to consume their outputs. Figure 4.16 also shows that the *VLC Binary Packer* is busy most of the time due to the large volume of output bitstream which causes the other upstream processors in the CAVLC encoder to stall on output during execution. Most of the processors stall on input which indicates that at some time the source processors are providing data at

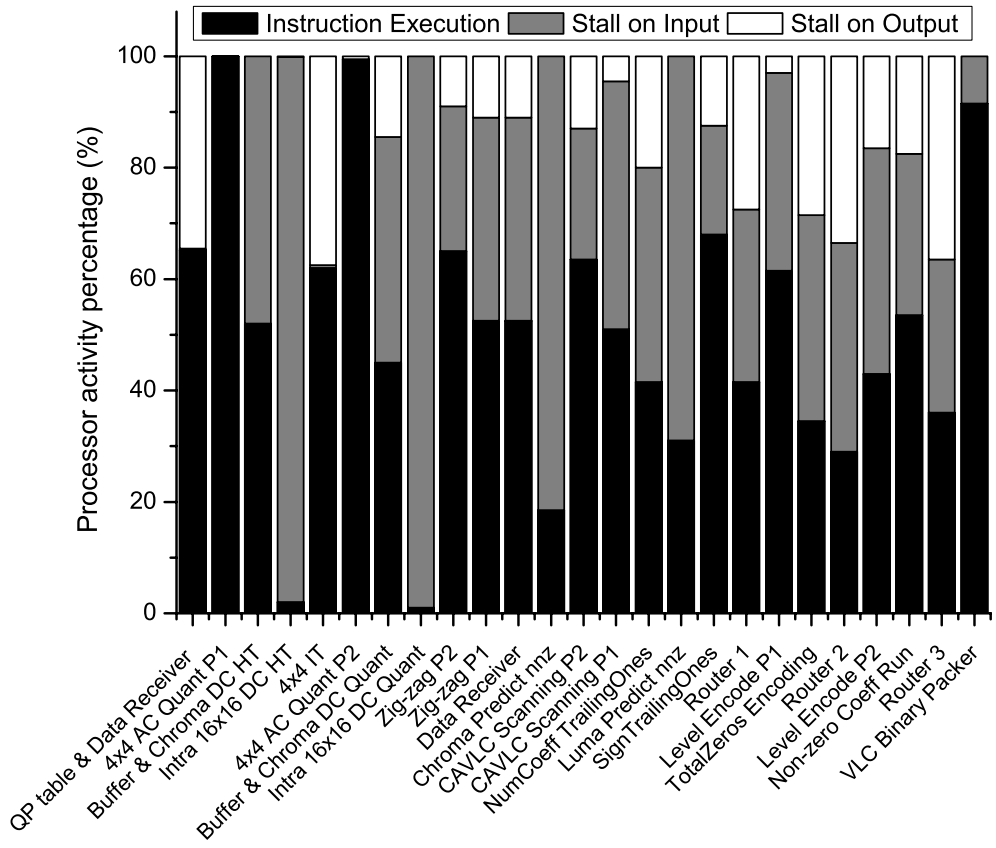



Figure 4.16: Processor activity of the residual encoder while encoding QCIF foreman at QP = 25.

a slower rate than the destination processor can consume it. The large amount of stall time in Figure 4.16 shows a large slack for most of the processors, which provides a potential to reduce the clock rate and supply voltage to increase energy efficiency.

### 4.5.3 Power Consumption Optimization

#### Power Estimation

One advantage of the target many-core system is that each processor its own oscillator. The clock can be totally halted when the processor stalls for a certain amount of time either because of input empty or output full. During a short stall, the clock can still be active which results in more power consumption than the case of a total standby with halted clock. The

overall activity of processors allows us to estimate the total average power by:

$$\begin{aligned} P_{Total} &= \sum_i P_{Exe,i} + \sum_i P_{Stall,i} + \sum_i P_{Standby,i} \\ &\quad + \sum_i P_{Comm,i} + P_{sharedmemory} \end{aligned} \tag{4.1}$$

where $P_{Exe,i}$, $P_{Stall,i}$, $P_{Standby,i}$ and $P_{Comm,i}$ represent the power consumption of computation execution, stalling with active clock, standby with halted clock and communication activities of the $i^{th}$ processor among 25 processors, respectively. $P_{sharedmemory}$ is the average power of the 16-KB shared memory. $P_{Exe,i}$, $P_{Stall,i}$, $P_{Standby,i}$ are estimated as follows:

$$\begin{aligned} P_{Exe,i} &= \alpha_i \cdot P_{ExeAvg} \\ P_{Stall,i} &= \beta_i \cdot P_{StallAvg} \\ P_{Standby,i} &= (1 - \alpha_i - \beta_i) \cdot P_{StandbyAvg} \end{aligned} \tag{4.2}$$

where $P_{ExeAvg}$, $P_{StallAvg}$ and $P_{StandbyAvg}$ are average power while the processor is 100% active in execution, stalling and standby (leakage only); $\alpha_i$, $\beta_i$ and $(1 - \alpha_i - \beta_i)$ are the percentage of execution, stall and standby activities of processor *i*, respectively. The communication power of processor *i* can be estimated as follows:

$$\begin{aligned} P_{Comm,i} &= \sum_j (\delta_{ij} \cdot P_{CommActive,L_j} \\ &\quad + P_{CommStandby,L_j}) \end{aligned} \tag{4.3}$$

where $\delta_{ij}$ is the communication active percentage of link *j*; $P_{CommActive,L_j}$ and $P_{CommStandby,L_j}$ are the average power consumed by a link with a length *L* while the link is 100% active and standby. Table 4.2 shows the measured average power consumption of various functions at 1.3 V and 1.2 GHz. We have included two types of communication link power since the length of the long-distance communication links in our application are no more than one tile. As shown in Table 4.2, all the components consume little standby power and the

Table 4.2: Power measured at 1.3 V and 1.2 GHz.

| Operation of | 100% Active (mW) | Stall (mW) | Standby (mW) |
|---|---|---|---|
| Processor | 62.0 | 31.0 | 0.13 |
| Shared Memory | 4.3 | NA | 0.11 |
| Nearest-neighbor comm. | 5.9 | NA | $\sim$0 |
| Long-distance comm. one tile | 12.1 | NA | $\sim$0 |

communication circuits consume nearly zero leakage due to their simplicity.

Based on the average cycles per macroblock data as we present in Figure 4.15, Table 4.3 lists the maximum frequencies to support realtime (30 fps) encoding of all the 8 test sequences. The processors only need to run as low as 15 MHz to encode QCIF foreman sequence at 30 fps. Among all the tests, the 1080p pedestrian area video sequence requires the highest frequency of 1032 MHz for real-time encoding. Based on equations 4.1, 4.2, 4.3 and Table 4.2, we can reasonably estimate the average power consumption of our residual encoder. We use the processor and communication activity data from the profiling of encoding all the 8 test sequences at $QP = 25$. Table 4.3 shows the power consumption of all the tests without voltage and frequency scaling which means all of the processors run at the same maximum frequencies and corresponding supply voltages. The QCIF foreman real-time encoding consumes only 4 mW and the power number increase proportionally with the frame size. The encoder consumes 115–121 mW for 720p HDTV tests at 30 fps and 433–544 mW for 1080p HDTV tests at 30 fps.

**Power Optimization**

The power dissipation of our encoder can be further reduced by adjusting the frequency and voltage of each processor. Based on the processor activity number, each processor has an optimal operating frequency so that the processors can be active as much as possible. By running at these optimal frequencies, the power wasted by stalling and standby activities of the processors is eliminated. As shown in Figure 4.16, in that case the two *AC 4x4 quant*

Table 4.3: Power consumption of residual video encoding running at 30 fps with and without static voltage and frequency scaling (VFS)

| Test | Frame Size | Max Freq. (MHz) | Power w/o VFS (mW) | Power w/ VFS (mW) | Power Change |
|---|---|---|---|---|---|
| Foreman | QCIF | 15 | 4.0 | 3.0 | –25% |
| Football | CIF | 45 | 9.1 | 7.1 | –22% |
| Soccer | 4CIF | 174 | 32 | 27 | –16% |
| Stockholm | 720p | 425 | 115 | 78 | –32% |
| Shields | 720p | 397 | 121 | 89 | –26% |
| Rush hour | 1080p | 939 | 433 | 271 | –37% |
| Pedestrian area | 1080p | 1032 | 544 | 347 | –36% |
| Blue sky | 1080p | 905 | 447 | 260 | –42% |

processors must run at the highest frequencies and the other processors can run at lower frequencies.

Our platform supports two global supply voltage grids $V_{ddHigh}$ and $V_{ddLow}$. The values of $V_{ddHigh}$ and $V_{ddLow}$ are variables for different test cases. The $V_{ddHigh}$ is chosen to support the maximum frequency based on the measured voltage frequency curve [74]. The $V_{ddHigh}$ is set to 1.15 V for all the three 1080p video tests shown in Table 4.3. Based on our simulation, the two AC quantization processors are set to run at $V_{ddHigh}$ for the three 1080p tests. The other processors can run at $V_{ddLow}$ or $V_{ddHigh}$ depending on their optimal operating frequency. If at $V_{ddLow}$ the processor can reach its optimal operating frequency, the supply voltage is set to $V_{ddLow}$; otherwise $V_{ddHigh}$ is chosen. To find the optimal $V_{ddLow}$ we changed $V_{ddLow}$ from $V_{ddHigh}$ down to 0.65 V and chose the $V_{ddLow}$ value which results in the minimum total power consumption.

Figure 4.17 shows the total power consumption corresponding to these $V_{ddLow}$ values for the Foreman video test. As shown in the figure, the optimal $V_{ddLow}$ is 1.05 V with total power of 582 mW for 1080p Foreman encoding at 30 fps, a reduction of 29.5% when compared with the previous case in which all processors run at 1.2 GHz and 1.3 V. Similarly, we can scale the voltage and frequency for the 720p encoder at 30 fps where $V_{ddHigh}$ is set at 0.90 V and the optimal $V_{ddLow}$ is 0.80 V, which reduces 61.3% of the power dissipation and results in 148 mW power in total.

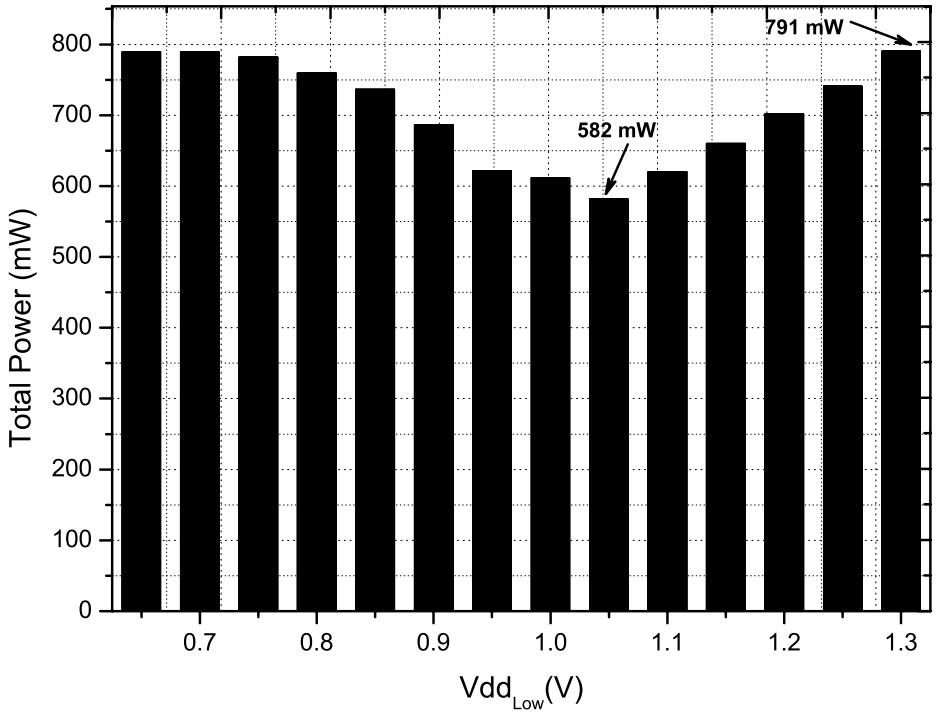
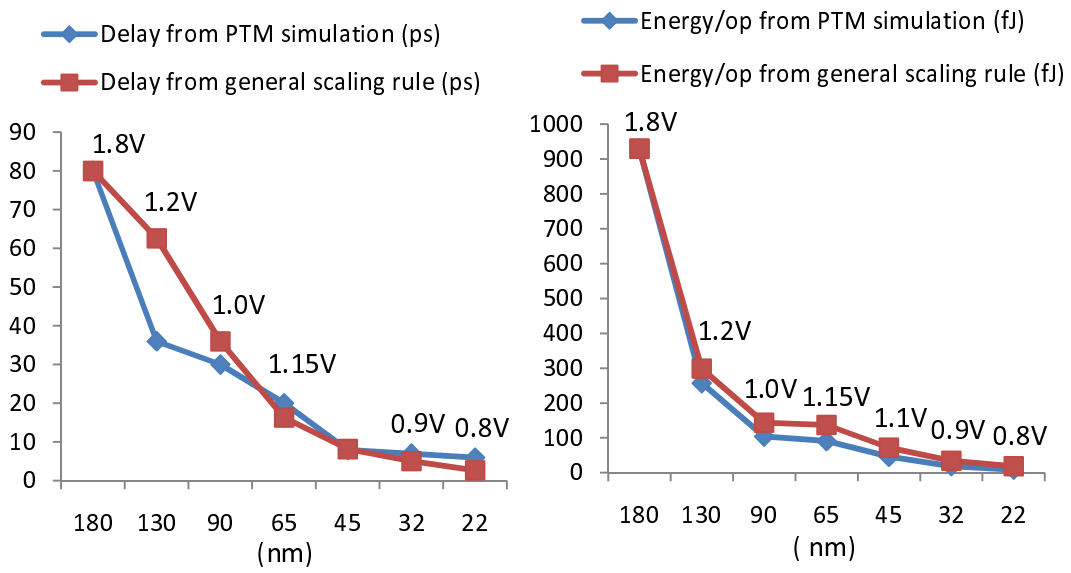



Figure 4.17: The total power consumption over various values of $V_{ddLow}$ (with $V_{ddHigh}$ fixed at 1.3 V) while processors running at their optimal frequency and encoding Foreman at QP=25. Each processor is set at one of these two voltages depending on its frequency.

Table 4.3 summarizes the estimated power consumption of encoding the eight video sequences at $QP = 25$ with voltage frequency scaling (VFS). As shown in Table 4.3, with VFS, the residual encoder only consumes 3 mW for QCIF foreman encoding at 30 fps. For the two 720p video tests, the encoder consumes 78–89 mW with VFS. On average, with $V_{ddHigh}$ and $V_{ddLow}$ at 0.85 V and 0.75 V, the encoder consumes 84 mW power dissipation for 720p video encoding at 30 fps—an average reduction of 29% compared with the design without VFS. For the three 1080p 30 fps video sequences, the encoder consumes 260–347 mW. On average, with $V_{ddHigh}$ and $V_{ddLow}$ at 1.15 V and 0.9 V, the encoder is capable of 1080p video encoding at 30 fps with 293 mW power dissipation—an average reduction of 38.4% compared with the design without VFS. The results demonstrate the effectiveness of voltage and frequency scaling for video applications with dynamic workloads. Another observation is that as frame size increases, the power savings increase with voltage and frequency scaling. This is because high-definition video encoding has more unbalanced




(a) Delay (b) Energy per operation

Figure 4.18: Delay and energy per operation of an inverter driving a fanout of 4 based on SPICE simulation using predictive technology model (PTM) [81]; the general scaling rule assumes a $v/s^2$ reduction in delay and a $1/(sv^2)$ reduction in energy/op where $s$ is the technology scaling factor and $v$ is the voltage scaling factor [82].

workloads among the encoding tasks, which provides more power-saving potentials for voltage and frequency scaling.

### 4.5.4 Performance Comparison

The H.264/AVC baseline encoder has been implemented on many DSP platforms. In order to fairly compare with other reference designs, we estimate the loading fraction of residual encoding in a full baseline encoder. Since this loading fraction is affected by many different variables such as processor architecture and test video sequences, we use a range to estimate the fraction number.

The CAVLC occupies 18.2% computation time of the full baseline encoder running on a general-purpose computer [83]. Our parallelized IT and Quant modules take around 56.2% computation time of CAVLC encoding. Since the other reported designs use VLIW, SIMD, or multiple-issue architectures which are very likely able to execute multiple instructions per cycle during the computation of IT and Quant, we estimate they double their

performance while computing these workloads. In this way, we estimate the IT and Quant take about 5.1% computation time of the full encoder. Summing up the two fractions, the residual encoder is estimated to take 23.3% computation time of a full encoder. We added a fluctuation ranging from $\pm 3\%$ to roughly estimate the test sequence variation which is observed in our JM encoding tests over various test sequences from QCIF to 1080p frame sizes. Thus, we estimate the residual encoder takes about 20.3% to 26.3% of a full baseline encoder.

For a fair comparison, all of the reference data are scaled to 65 nm technology at a supply voltage of 1.15 V. We use a technology scaling rule justified by SPICE simulation of an inverter driving a fan out of 4 under different technology nodes and supply voltages with prediction technology model (PTM) [81] as shown in Figure 4.18. We use the metrics of throughput (Mpixel/s), throughput per area ((Mpixel/s)/mm$^2$), energy per pixel (nJ/pixel) to compare the throughput, hardware efficiency and energy efficiency of each design.

Based on the loading fraction and technology scaling rule, we estimate the residual encoder performance of published software H.264/AVC baseline encoders on two DSP platforms and two hybrid multi-core architectures as shown in Table 4.4. Since the proposed residual encoder on AsAP is configurable and programmable, we include the performance data of our design encoding 1080p, 720p and CIF at 30 fps at different supply voltages as shown in in Table 4.4. The energy per pixel of AsAP reduces as we reduce the frame size and supply voltages. A reduction of 36% and 52% energy per pixel are achieved for 720p and CIF video encoding compared to 1080p encoding.

For a fair comparison, we only compare the other designs with AsAP while encoding 1080p at 30 fps because the other results are scaled to 65 nm and 1.15 V. As shown in Table 4.4, compared with the encoder on the TI DSP C642, the proposed residual encoder on AsAP has 2.9–3.7 times higher throughput, 11.2–15 times higher throughput per chip area and 4.5–5.8 times smaller energy per pixel. Compared with ADSP BF562 DSP, our design has 2.3–3.0 times higher throughput and 5.6–7.2 times smaller energy per pixel. The

| Platform | Arch. | Tech. (nm) | Vdd (V) | Area ($mm^2$) | Max Freq (MHz) | Power (mW) | Throughput | Esti. Resi.[a] Throughput (Mpixel/s) | Est. Resi.[a] Energy (nJ/pixel) | Other results scaled to 65 nm & 1.15 V | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | Throughput (Mpixel/s) | Throughput/Area ((Mpixel/s)/$mm^2$) | Energy (nJ/pixel) |
| TI C642 [84] | 8-way VLIW | 130 | 1.2 | 72 | 600 | 718 | CIF@24fps | 9.3–12.0 | 59.8–77.2 | 16.7–21.6 | 0.9–1.2 | 21.2–27.4 |
| ADSP BF561 [85] | Dual-core DSP | 130 | 1.2 | NA | 600 | 1110 | CIF@30fps | 11.6–15.0 | 74–95.7 | 20.9–27.0 | NA | 26.2–33.9 |
| Cell [86] | CPU + SIMD PE | 90 | 1 | 221 | 3200 | NA | **1080p@31fps** | **244–317** | NA | **366–476** | 2.8–3.2 | NA |
| SODA[b] customized for H.264 [87] | CPU + SIMD PE | 90 | 1 | 14.29 | 300 | 68 | CIF@30fps | 11.6–15.0 | 4.5–5.9 | 17.4–22.5 | 2.3–3.0 | 3.9–5.2 |
| Intel P8400[c] [88] | Dual-core CPU | 45 | 1.1 | 107 | 2260 | 12,500[d] | 1080p@12fps | 25 | 500 | 13.2 | 0.06 | 437.9 |
| ASIC[e][89] (ISSCC2007) | *ASIC* | *130* | *1.2* | *4.0* | *108* | *36.6* | *720p@30fps* | *27.7* | *1.3* | *106.1* | *106.1* | *0.60* |
| **This work AsAP**[f] | Array (25 cores) | 65 | 1.15/0.9 | **4.6** | **959** | **293** | 1080p@30fps | 62.2 | **4.7** | 62.2 | **13.5** | **4.7** |
| | | | 0.85/0.75 | **4.6** | **411** | **84** | **720p@30fps** | 27.6 | **3.0** | 27.6 | 6.0 | **3.0** |
| | | | 0.675/0.675 | **4.6** | **45** | **7.1** | **CIF@30fps** | 3.0 | **2.3** | 3.0 | 0.65 | **2.3** |

[a] The residual encoding throughput is estimated based on a loading factor of 20.3%–26.3% of a full baseline encoder.
[b] SODA is not fabricated and data are from synthesis results [87].
[c] Measured results by implementing the same residual encoder on Thinkpad T400 Core 2 Duo PC.
[d] The P8400's typical power is not available, so 50% of TDP (25W) is used based on benchmark data of a general-purpose processor [90].
[e] The residual encoder is estimated to be one sixth the total chip area based on the die photo and the power is estimated to be 20% of the total power.
[f] The AsAP's area includes 25 cores and one 16-KB shared memory. Three sets of supply voltages are used for 1080p, 720p and CIF video encoding separately.

Table 4.4: Comparison of residual encoder on different software platforms and ASICs; the original published data are included under different technology nodes and supply voltages; For comparison, data are scaled to 65 nm technology with a supply voltage 1.15 V assuming a $1/s^2$ reduction in area; throughput and energy are scaled based on a scaling rule justified by the SPICE simulation shown in Figure 4.18.

IBM cell processor is a heterogeneous multi-core architecture for high-end gaming and multimedia processing [91]. The reference design on Cell has 5.9 to 7 times higher scaled throughput than our design at a cost of 4.2 to 4.8 lower area efficiency than AsAP. The Cell processor power number is not available though AsAP should have far higher energy efficiency due to area alone. The customized SODA is specially optimized for H.264 by introducing flexible SIMD width, diagonal memory organization and special fused operation instructions [87]. Compared to the customized SODA, our implementation achieves 2.8 to 3.6 times higher throughput and 4.5 to 5.9 times higher area efficiency. AsAP has similar energy efficiency compared to the SODA customized for H.264. SODA has not been fabricated and both area and power data are from synthesis results [87].

We also implemented the same residual encoder written in sequential C and compiled it with Intel C++ Compiler 9.1 on a state-of-the-art Intel Core 2 Duo P8400 computer running Windows XP SP2 with 3G Bytes DDR3 RAM. To be fair, we doubled the performance estimation of our sequential implementation based on the fact the encoder could be potentially parallelized at the thread-level on the dual-core processor [59]. As shown in Table 4.4, the throughput of our design is around 4.7 times the scaled throughput of the design running on the P8400. Our results show a state-of-the-art general-purpose processor can not meet real-time 1080p encoding requirement with around two orders of magnitude smaller throughput per area and around 93 times higher energy per pixel compared with our design on AsAP.

For a complete comparison, we also estimate the area and power consumption of a hardware residual encoder based on a 720p H.264 baseline encoder chip fabricated at 130 nm CMOS [89]. Based on the die photo, the area of the residual encoder is estimated to occupy one sixth the total chip area. The power of the residual encoder is related to both workload (23.3%) and the chip area. Thus, a medium value (20%) is used to estimate the total power consumption of the residual encoder. As expected, ASIC achieves higher area and energy efficiency — 7.9 times higher area efficiency and 7.8 times higher energy efficiency. However, the efficiency of hard-wired ASICs come at the cost of little flexibility.

## 4.6 Conclusion

We have implemented a high-performance parallel H.264/AVC baseline residual encoder on a fine-grained many-core system. The encoder is composed of integer transform, quantization and CAVLC blocks. The 25-processor residual encoder is the first software implementation on a fine-grained many-core system that supports realtime 1080p HDTV encoding to the best of our knowledge. We exploited data and task level parallelism in the H.264/AVC algorithms at the fine-grained block level and utilized the benefits of the GALS architecture to reduce power dissipation based on the workload of each processor. The design achieves higher throughput, much higher throughput per chip area, and much lower energy per pixel than the exact same encoder implemented on a general-purpose multiprocessor. It also compares very well with published implementations on programmable DSP processors, thus demonstrating the great promise of fine-grained many-core processor arrays for use in video encoding.

# Chapter 5

# Application-Driven Processor Shape and Topology Design

## 5.1 Introduction

For many-core processors, long inter-processor communication links dissipate significant power that does not directly contribute to the workload processing [92]. Thus, many-core processors that utilize scalable interconnects and aovid global wires normally can attain higher peorformance.

Network-on-Chip (NoC) approaches are used to connect large numbers of processors on a single-chip because they are more efficient than less scalable methods such as global shared buses. There exist many design alternatives for NoC architectures which differ mainly in switching policy, topology and routing algorithms. Most proposed NoC architectures are based on dynamic packet switch routing and some are based on static configurable circuit-switch interconnection which has smaller area, lower power dissipation and lower complexity while trading off routing flexibility [93].

Network topologies define how nodes are placed and connected, affecting the latency, throughput, area and power of a network. Due to its simplicity and the fact that processor

tiles are traditionally square or rectangular, the nearest-neighbor 2D mesh topology is a natural solution for both dynamic and static on-chip communication architectures. However, efficiently mapping applications can be a challenge for cases that require communication between processors that are not adjacent on the 2D mesh. This condition could require processors to forward data for static interconnection architectures, and intermediate routers for dynamic router-based NoCs. The power consumption and communication latency also increase as the number of routing processors or routers between two communicating cores increase. There exist other common topologies for NoCs such as 2D torus, fat tree, octagon and higher dimensional meshes and tori which provide higher routing capability and communication bandwidth with costs of higher wire density and longer global wires [94]. Furthermore, these topologies present significant challenges for many-core physical implementations especially with the number of cores per die expected to soon reach thousands and more.

For many applications mapped onto homogeneous chip multiprocessors, communication between processors is often largely localized [95, 96], which may result in local mapping congestion; an increase of local connectivity can ease such congestion. This motivates us to propose new topologies with increased local connectivity while keeping much of the simplicity of a mesh-based topology.

Many-core NoC topology design has a strong impact on application performance, physical design time and application mapping effort. This work proposes regular and scalable topologies and tile shapes for dense interconnection of many-core arrays which result in an overall application processor with fewer cores and a lower total communication length.

The main contributions of this chapter can be summarized as follows:

1. Seven NoC topologies are proposed and compared to the common 2D mesh including two 8-neighbor topologies, two 5-neighbor topologies and three 6-neighbor topologies. Three of them utilize hexagonal-shaped or 5-sided “house-shaped” processor tiles.

2. A complete functional H.264/AVC residual encoder and an 802.11a/g OFDM baseband receiver are mapped onto all topologies for realistic comparisons.
3. Commonly available commercial CAD tools are used to implement tiled CMPs for all proposed topologies. All seven topologies including the hexagonal and house-shaped processor tiles are physically implemented in 65 nm CMOS using standard cells and Manhattan-style wires without full-custom layout. The final layouts are all DRC and LVS clean.

The remainder of this chapter is organized as follows. Section 5.2 reviews related work. Section 5.3 describes and evaluates the proposed inter-processor communication topologies. Section 5.4 presents mapping of two applications onto a 2D mesh and all proposed topologies. Section 5.5 describes the physical design flow and the approach to implement the non-rectangular processor tiles. Section 5.6 presents the chip implementation results and section 5.7 concludes this chapter.

## 5.2 Related Work

Many topologies have been used for on-chip inter-processor communication, such as buses, meshes, tori, binary trees, octagons, hierarchical buses and custom topologies for specific applications [94]. Many high radix topologies have been proposed to minimize hop count at the cost of higher routing complexity and possibly higher energy consumption [97, 98]. The low complexity 2D mesh has been used by most fabricated many-core systems including RAW [66], AsAP [11, 72], Intel 80-core [99], TILE64 [47], AsAP2 [73, 74] and Intel 48-core Single-Chip Cloud Computer (SCC) [100].

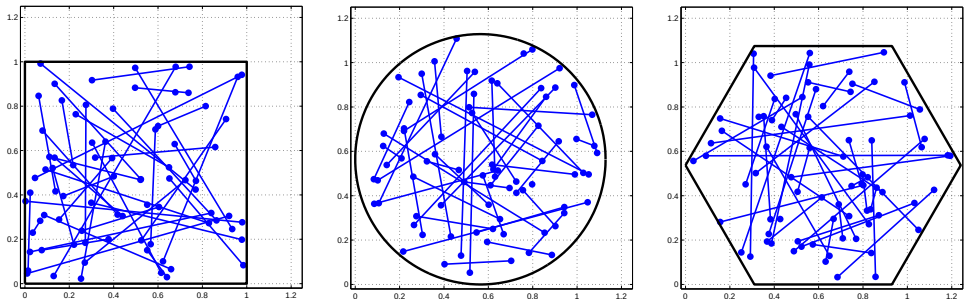
Prior work has been reported using hexagonal interconnections for on-chip wire routing and off-chip multiprocessor communication. Chen et al. [101] propose Y architecture for on-chip interconnections and show that it can increase communication throughput by 20.6% over the 2D mesh with Manhattan-style wires. Zhou et al. [102] propose hierarchi-

cal three-way interconnection, Y tree architecture, for hexagonal processors. These two papers only theoretically propose hexagonal interconnection architectures and showcase the throughput benefit only if *non-Manhattan* style wires are used. Shin [103] proposes a hexagonal mesh for the interconnection of multiple processors in a system, which has been demonstrated to have higher communication performance and robustness than other topologies. Furthermore, Decayeux and Seme proposed a 3D hexagonal network as an extension of 2D hexagonal networks [104]. As mentioned before, such off-chip hexagonal networks are used to connect computation nodes, which is different from our proposed on-chip hexagonal-shaped processor tiling.

Becker et al. [105] developed a hexagonal Field-programmable Analog Array in a 0.13 μm CMOS technology. The basic building block is a hexagonal analog circuit block which communicates with six neighbors. Extension to a many-core processor is similar in topology, but very different in terms of impact on tile area and total application interconnect. Malony studies the two-dimensional regular processor arrays which are geometrically defined based on nearest-neighbor connections and space-filling properties [106]. He theoretically proves the hexagonal array is the most efficient topology in emulating other topologies by analyzing the geometric characteristics.

## 5.3 Processor Shapes and Topologies

In this section, various topologies are proposed, where several use non-rectangular processor shapes for compact tiling. The proposed topologies avoid long global wires and increase routing capability and communication bandwidth compared with the 2D mesh. The worst-case communication distance for four basic communication patterns and the maximum interconnect wire delay for different processor tiles are used to evaluate all topologies.



(a) Square (b) Circle (c) Hexagon

Figure 5.1: Example tiles of constant area with random uniformly-distributed wire endpoints.

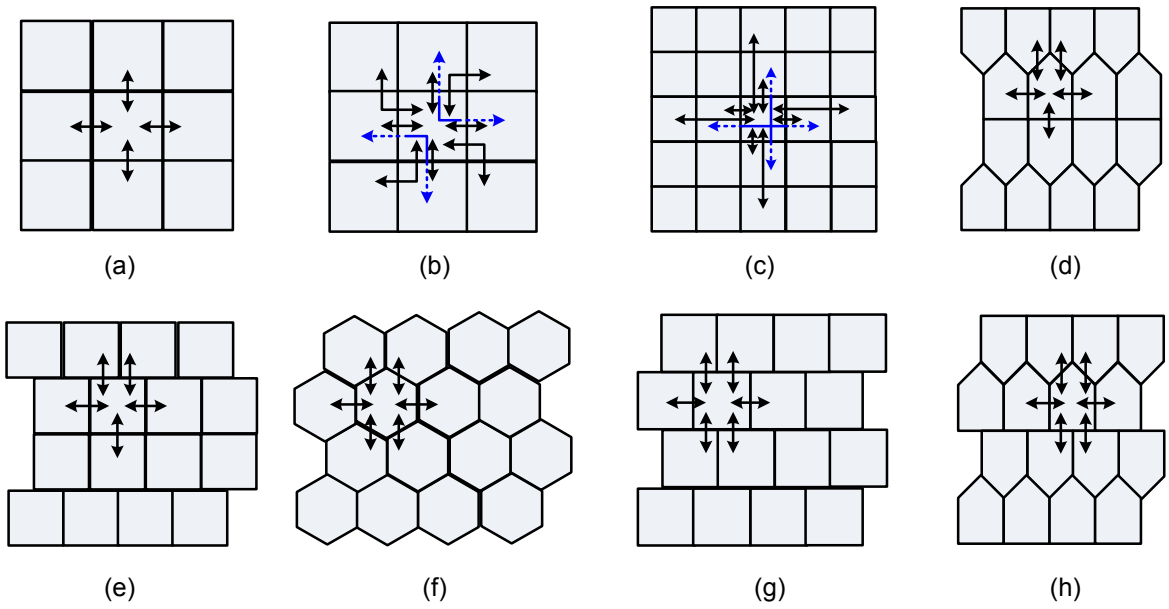



Figure 5.2: The (a) baseline 2D mesh (4-4 Rect) and seven proposed topology/shape combinations: (b) 8-8 Rect, (c) 8-4 Rect, (d) 5-5 House, (e) 5-5 Rect Alt. Offset, (f) 6-6 Hex, (g) 6-6 Rect Offset, and (h) 6-6 House Offset. (Designs are named using: the total number of interconnection links, the number of nearest-neighbor interconnection links, and the processor's shape.)

### 5.3.1 Processor Tile Shapes

To the best of our knowledge, all previously-fabricated VLSI processors have been of a rectangular shape, often nearly square. As illustrated in Figure 5.1(a)(b), it stands to reason that a circular shape would allow shorter wires for a given netlist, resulting in smaller area and lower wire capacitance which would result in higher speeds and lower energy per operation. A simple experiment with ideal shapes and one million randomly-placed wires yields a 2.2% reduction in total wire length for a circular tile compared to a square tile. On the negative side, it is clear that circles do not pack together without wasted space between tiles. On the positive side, circles pack with *six* neighbors while rectangles obviously have only four. It is reasonable to expect a rectangular tile to have longer wires on average compared to a square tile.

In contrast to the circle, the hexagonal shape *does* pack efficiently without gaps between tiles and it retains the 6-nearest-neighbor property. The same wiring experiment was run for a hexagonal tile and it resulted in a 1.8% reduction in total wire length compared to the square tile.

A reduction in total wire length yields a pure benefit in area, energy and delay for processor tile design. The inclusion of common rectangular blocks such as memory arrays in a processor tile increases routing congestion but is shown in Section 5.6 to be tolerable. In addition, we demonstrate that Manhattan-style wire routing is fully compatible with non-rectangular tile shapes.

### 5.3.2 The Proposed Topologies

The eight different topologies in combination with processor tile shapes are shown in Figure 5.2. Switch fabrics are assumed to reside inside each processor tile. The well-known 2D mesh in Figure 5.2(a) is used as the baseline topology for comparison. All topologies are named by: 1) the total number of direct interconnection links, 2) the number of nearest-

neighbor interconnection links, where nearest neighbors are defined as directly connected processors that touch at the edge or the vertices. and 3) the processor's shape. For example, the baseline 2D mesh is named *4-4 Rect* where tiles are rect-shaped and connected by four links, all of which are nearest-neighbor interconnect links.

The next logical extension of the 2D mesh is to include the four diagonal processors in an 8-neighbor arrangement named *8-8 Rect* as shown in Figure 5.2(b) where each rect tile can directly communicate with 8 neighbors. This approach has increased routing congestion in the tile corners due to the four (uni-directional) links that pass through each corner (the dashed lines in Figure 5.2(b)).

The third topology is an 8-neighbor mesh (*8-4 Rect*) as shown in Figure 5.2(c) where the baseline 2D mesh is augmented with direct connections with processors two tiles away. In this case, the "pass through" routes are not just in the corners, but pass through the entire tile.

Figure 5.2(d) shows a 5 nearest-neighbor topology (*5-5 House*) where each tile is a "house-shaped" pentagon. There are various house shapes and the center-to-center Euclidean distances between a tile's center and its five neighbors are not equal. However, the center of a house-shaped tile can be chosen so that the Euclidean distances from the center to all five vertices are equal, which yields only one type of house-shaped tile where the rectangular shape at the bottom is square. If the square shape has an edge length of $w$, the center-to-center distance for three of the five connections is $w$ and the other two connections have a length of $w * \sqrt{(2+\sqrt{2})/2}$.

Figure 5.2(e) shows an alternative 5-neighbor topology (*5-5 Rect Alt. Offset*) where every other row of rect tiles are offset. The *5-5 Rect Alt. Offset* has the same interconnection topology as the *5-5 House*. All processors are square-shaped with an edge length of $w$. The center-to-center Euclidean distance between two processor tiles can be either $w$ (if tiles are aligned) or $\sqrt{5}/2 * w$ (if tiles are in an offset position). This topology has the advantage of a regular processor shape while achieving the same routing capability as the house-shaped

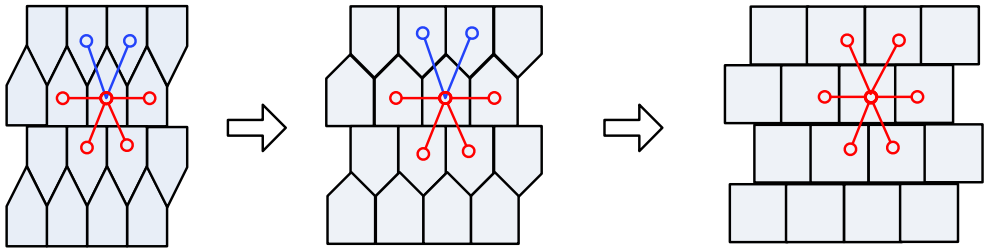


Figure 5.3: A spectrum of 6-neighbor topologies with offset row house-shaped tiles which differ in the area of the triangle roof of the house shape.

tile topology.

Our sixth proposed interconnect topology is the 6-nearest-neighbor array using hexagonal-shaped processor tiles as shown in Figure 5.2(f). The processor center-to-center Euclidean distance is $\sqrt{3} * w$ if the length of the hexagon edge is $w$. The hexagonal grid is commonly used in mobile wireless networks due to its desirable feature of approximating circular antenna radiation patterns and its optimal characteristic of six nearest neighbors. The symmetry and space-filling property make the hexagonal processor tile topology an attractive design option for many-core processor tiles.

Figure 5.2(g) shows our seventh topology named *6-6 Rect Offset* where every row of tiles is offset so that each tile has 6 nearest neighbors. For tiles with height $h$ and width $w$, the center-to-center distance in the horizontal direction is clearly $w$. For adjacent tiles in the row above and below, the center-to-center Euclidean distance is $\sqrt{w^2/4 + h^2}$. Thus, if we set $w = \sqrt{w^2/4 + h^2}$, or $h = \sqrt{3}/2 * w$, then all six neighbors will reside at equal center-to-center Euclidean distances.

Figure 5.2(h) shows the eighth topology (*6-6 House Offset*) where every neighboring row of house-shaped tiles are offset so that each tile has 6 neighbors. In fact, as shown in Figure 5.3, there are a spectrum of topologies that fall into this category where the triangle roof of the house-shaped tile can have varying area. However, there is no geometrically optimal topology with six equal-Euclidean-distance neighbors. If the area of the roof triangle is 0, it becomes the *6-6 Rect Offset* topology which has the advantage of equal center-to-center Euclidean distances for all six neighboring tiles as shown in Figure 5.2(g).

Table 5.1: Euclidean and Manhattan link lengths for all topologies with one unit of length equal to the square root of the area which is one for all topologies and shapes

| **Topology** | **Nearest-neighbor Link** | | | **Longer Link** | | |
|---|---|---|---|---|---|---|
| | Num. | E. Dis. | M. Dis. | Num. | E. Dis. | M. Dis. |
| 4-4 Rect | 4 | 1.00 | 1.00 | 0 | – | – |
| 8-8 Rect | 4 | 1.00 | 1.00 | 4 | 1.41 | 2.00 |
| 8-4 Rect | 4 | 1.00 | 1.00 | 4 | 2.00 | 2.00 |
| 5-5 House | 3 | 0.95 | 0.95 | 2 | 1.24 | 1.51 |
| 5-5 Rect Alt. Offset | 3 | 1.00 | 1.00 | 2 | 1.12 | 1.50 |
| 6-6 Hex | 6(2)* | 1.07 | 1.07 | 0(4)* | – | 1.47 |
| 6-6 Rect Offset | 6(2)* | 1.07 | 1.07 | 0(4)* | – | 1.46 |

* The *6-6 Hex* and *6-6 Rect Offset* have six nearest-neighbor links using Euclidean wires. However, the two topologies have two nearest-neighbor links and four longer links using Manhattan-style wires.

Therefore, we consider only the *6-6 Rect Offset* for this type of topology in the following sections.

The center-to-center distance can be used to represent the communication link length between two processor tiles. Table 5.1 shows the number of different types of communication links and the corresponding link length for all topologies. For comparison purpose, the link lengths are calculated based on both Euclidean and Manhattan rule. As shown in Table 5.1, if Euclidean rule is used, the *4-4 Rect*, *6-6 Hex* and *6-6 Rect Offset* have only one type of communication link due to equal center-to-center Euclidean distance. The *8-8 Rect*, *8-4 Rect*, *5-5 House* and *5-5 Rect Offset* topologies have two types of links due to the unequal center-to-center Euclidean distance between processor tiles. If Manhattan rule is used, all topologies have two types of link except the *4-4 Rect* 2D mesh. The *6-6 Hex* and *6-6 Rect Offset* have two short links and four long links instead.

Due to limitations of current wafer sawing technologies, chips from round wafers are traditionally square or rectangular. In fact, the opportunities and limitations of non-rectangular processors on a chip are analogous to non-rectangular chips on a wafer. For the case of a rectangular chip composed of non-rectangular processors, there are areas on the periphery of the chip in which processors cannot be placed for the topologies shown in

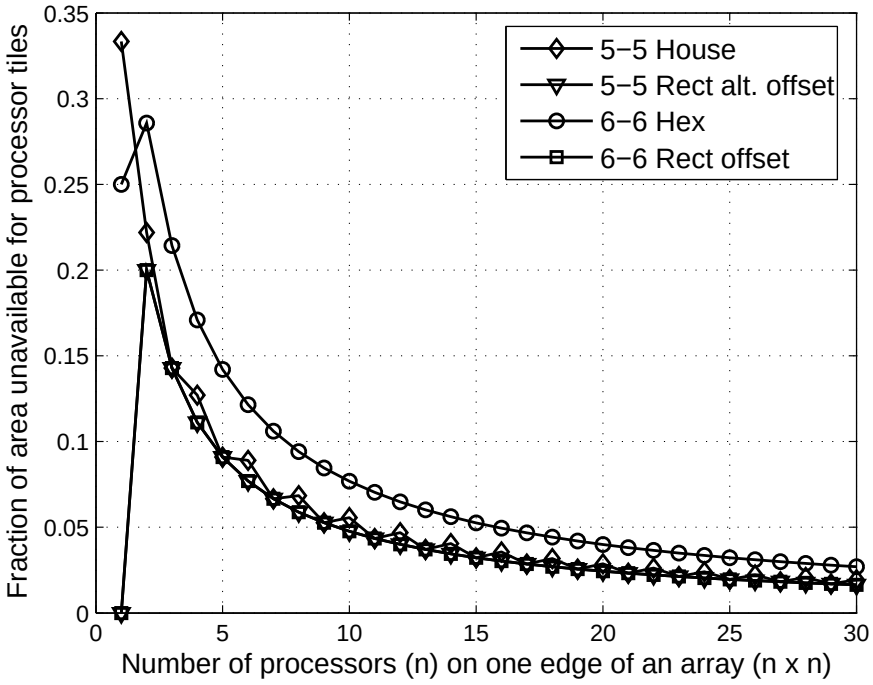
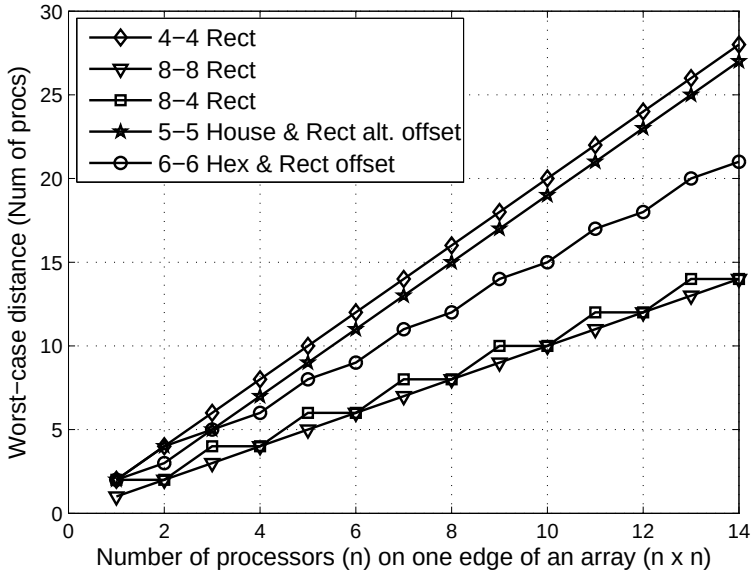



Figure 5.4: Fraction of area unavailable for processor tiles in a non-mesh array ($n \times n$) for type d–g in Figure 5.2: 5-5 House, 5-5 Rect Alt. Offset, 6-6 Hex and 6-6 Rect Offset, respectively.

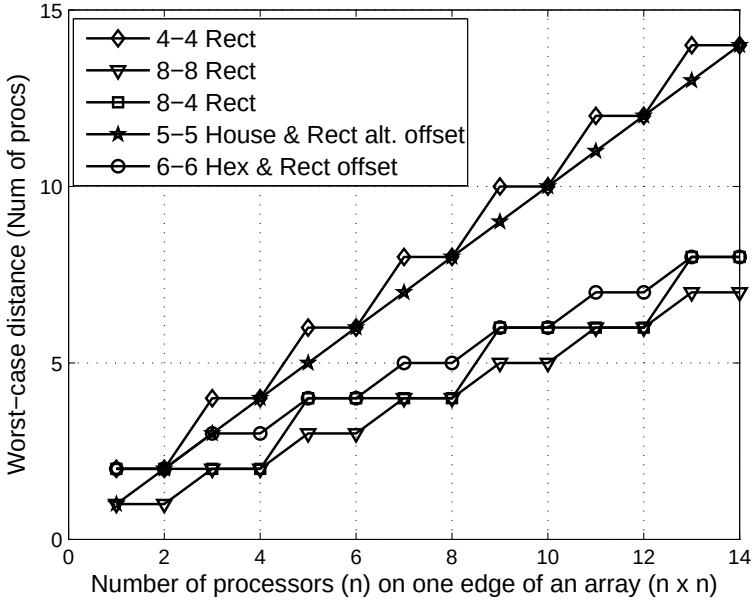
Figure 5.2(d), (e), (f), (g) and (h). Figure 5.4 shows the percentage of unavailable area for the four topologies with varying processor array sizes. If the processor array size is larger than 30 by 30, this area overhead becomes less than 2.7% of the total chip area for the hexagonal-shaped tile array and 2.0% for the house-shaped tile array. The overhead area for type (e) and (g) is less than 1.7% of the total chip area. In practice, these areas could be filled with useful chip components such as decoupling capacitors, or portions of hardware accelerators, memory modules, I/O circuits or power conversion circuits.

### 5.3.3 Performance Evaluation

This section analyzes the performance of the proposed seven topologies by using the worst-case communication distance of four basic communication patterns which include one-to-one communication (in which two processors at opposite corners of the processor




(a)




(b)

Figure 5.5: Comparisons of the worst-case communication distance across a processor array ($n \times n$) for different topologies where the number of input ports of each processor is equal to the number of interconnection links for four basic communication patterns: (a) one-to-one, one-to-all and all-to-all, and (b) all-to-one.

array communicate with each other), one-to-all broadcast (in which one corner processor broadcasts data to all the other processors), all-to-one communication (in which all processors communicate with the processor in the middle of the array) and all-to-all communication (in which every processor communicates with all other processors). All real application communication patterns can be a combination of these four communication patterns.

The number of neighboring inter-processor communication links and the number of input ports determine the local communication capability of a topology. The input port of a processor refers to the communication interface including buffers and related circuits. The number of input ports can be less than the number of neighboring interconnection links of one processor.

As shown in Figure 5.2, depending on the number of interconnection links, the topologies require a different number of input ports to make maximal use of nearest-neighbor interconnections. The baseline *4-4 Rect* mesh requires four input ports and the two 8-neighbor Rect topologies require eight input ports. The house-shaped tile and hexagonal-shaped tile topology require five ports and six ports, respectively. The increase of the number of input ports incurs significant hardware overhead in terms of buffers and related communication circuitry. However, the number of input ports into the local processor can be less than the number of neighboring interconnections, in which case processors are capable of talking to all connected neighboring processors but not at the same time. The following subsections discuss both the case with the same number of input ports as the neighboring interconnections and the case with a limitation of two input ports for all proposed topologies.

**Varying number of input ports**

In tiled CMPs, if processors have varying numbers of input ports, the communication can be distributed in multiple directions. Thus, the worst-case communication distance is

shorter for those topologies with more input port processors and nearest-neighbor interconnections.

For one topology, the worst-case communication distance of one-to-one, one-to-all and all-to-all communications are the same. As for different topologies, Figure 5.5(a) shows that the *4-4 Rect*, *5-5 House* and *5-5 Rect Alt. Offset* have similar worst-case communication distances which are approximately linearly proportional to the size of the array. The worst-case communication distances of the *6-6 Hex* and *6-6 Rect Offset* topologies are shorter than those of the *4-4 Rect*, *5-5 House* and *5-5 Rect Alt. Offset* topologies. The two 8-neighbor Rect meshes have the shortest worst-case communication distances because of more interconnection links.

Figure 5.5(b) shows the all-to-one worst-case communication distances for all seven topologies. The performance trends are similar to the one-to-one, one-to-all and all-to-all communication cases. The *6-6 Hex* and *6-6 Rect Offset* topologies show very close performance to the two 8-neighbor Rect meshes which have two more sets of communication links.

**Two input ports**

By limiting the number of input ports of the local processor to two, all topologies have the same one-to-one and one-to-all performance as shown in Figure 5.5(a) where varying numbers of ports are used. This means for one-to-one and one-to-all communication, two input ports are enough for all seven topologies and an increase of input ports does not improve the results. For all-to-one and all-to-all communication patterns, the topologies with more links have the same worst-case communication distance as the topologies with fewer links if one processor has only two input ports as shown in Figure 5.6. This shows that a reduction of input ports decreases the performance of the topologies with more links in the case of all-to-one and all-to-all communications.

For simple single-issue processors that normally consume no more than two operands

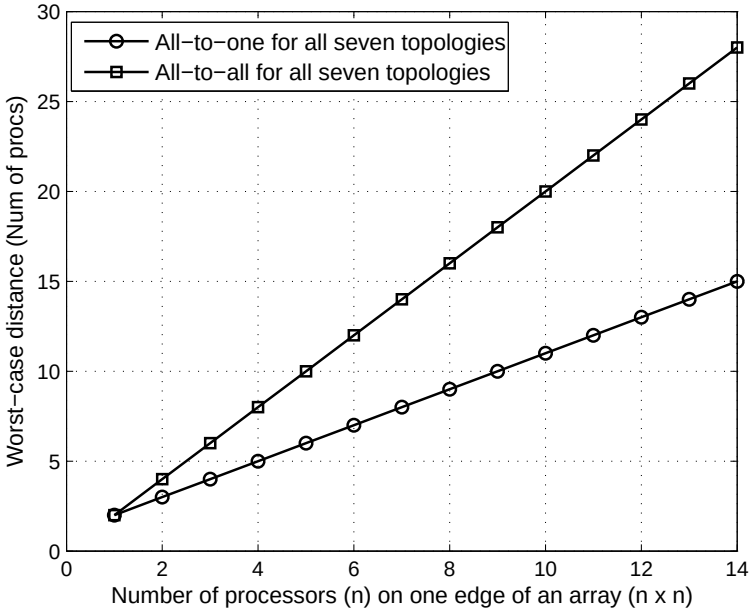



Figure 5.6: Comparisons of the worst-case communication distance across a processor array ($n \times n$) with a limitation of two input ports for each processor.

per clock cycle, adding more than two input ports may not have the benefits as shown in the worst-case analyses with varying number of input ports. Of course there may be benefits if the processor uses complex instruction styles which process more than two operands per cycle. For many cases, it is attractive to use two input ports for all proposed topologies, in which case the hardware overhead is minimized without affecting the communication performance too much. This limitation is justified by mapping two complex applications onto the proposed topologies in Section 5.4.

### 5.3.4 Interconnect Wire Delay

All discussed topologies permit an easy tiling of processors for dense on-chip networks without very long global wires. The two topologies in Figure 5.2(b) and Figure 5.2(c) have a maximum global interconnect link length no more than the dimension of one processor tile. There are no global wires for 2D mesh, house-shaped and hexagonal-shaped tile topologies. The actual delay of local interconnect wires, which is proportional to the

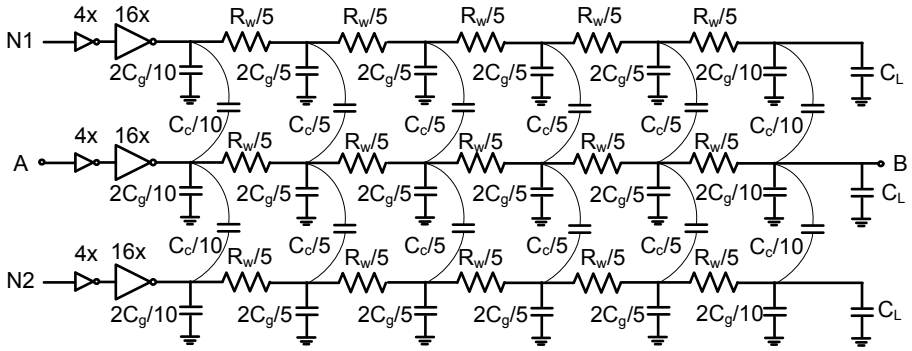



Figure 5.7: The $\Pi 5$ lumped RC circuit model used to simulate the wire delay for different shaped processor tiles considering crosstalk effects between the main wire transmitting signal from A to B and two adjacent wires N1 and N2. $R_w$, $C_g$ and $C_c$ are metal wire resistances, ground capacitances and coupling capacitances from adjacent intra-layer wires, respectively.

Table 5.2: Interconnect link wire length, delay and length and delay skew for square-shaped, house-shaped and hexagonal-shaped processor tiles with different sizes at 32 nm CMOS technology

| | Proc. tile area = 0.04 mm$^2$ | | | Proc. tile area = 4.0 mm$^2$ | | | Proc. tile area = 36 mm$^2$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Square | House | Hex | Square | House | Hex | Square | House | Hex |
| Max. link length (mm) | 0.30 | 0.35 | 0.32 | 3.00 | 3.53 | 3.16 | 9.00 | 10.59 | 9.48 |
| Max. link length diff. (mm) | 0.20 | 0.26 | 0.29 | 2.00 | 2.58 | 2.93 | 6.00 | 7.74 | 8.79 |
| Max. link wire delay (ps) | 28.6 | 34.7 | 31.1 | 338.0 | 391.7 | 347.6 | 1014.0 | 1186.0 | 1060.0 |
| Max. link wire delay skew (ps) | 22.9 | 24.9 | 27.5 | 217.8 | 282.6 | 324.8 | 676.0 | 866.6 | 983.2 |

size of processor tiles, depends on the physical position of the switch fabrics inside the local processor tile. Table 5.2 shows estimates of the maximum interconnect link lengths and maximum link length differences for the square-shaped, house-shaped and hexagonal-shaped processor tiles based on three nominal processor tile sizes 0.04 mm$^2$, 4 mm$^2$ and 36 mm$^2$ which approximate the scaled area of one processor tile in 32 nm CMOS for AsAP2 [73, 74], TI C64x DSP [76] and the Intel Sandy Bridge processor [107], respectively. All wire lengths are calculated based on the Manhattan-style wiring.

Figure 5.7 shows the $\Pi 5$ lumped RC model used to simulate wire delay while considering effects of crosstalk noise. As a common case, the wires are assumed to be in an intermediate layer which incurs both ground and coupling capacitances depending on the metal

wire dimensions (space, width, thickness, length) and inter-layer dielectric [108]. The simulation is based on HSPICE utilizing the device model from 32 nm CMOS Predictable Technology Model (PTM) [109]. The wire dimensions used for simulation are derived from International Technology Roadmap for Semiconductors (ITRS) [110] reports. The metal resistance, ground and coupling capacitance values ($R_w$, $C_g$ and $C_c$ in Figure 5.7) are calculated by PTM online interconnect tool. The center victim wire delay is measured from input A to output B including the buffer composed of two FO4 inverters. Based on the single buffered wire delay data (wire length from 0.1 mm$^2$ to 2 mm$^2$ with 0.1 mm$^2$ interval), long wires can be optimally segmented, which provides a more realistic delay estimation. As shown in Table 5.2, the delay data are based on the worst-case scenario where the signal on the center victim wire moves in the opposite direction of its aggressor neighbors N1 and N2.

Table 5.2 shows the interconnect wire delay and delay skew for square-shaped, house-shaped and hexagonal-shaped processor tiles with the three sizes. For the 0.04 mm$^2$ small processor tile running at a 2 GHz clock frequency, the maximum interconnect wire delay for all processor shapes ranges from 5.7% to 6.9% of one clock cycle. For the 4 mm$^2$ medium sized processor tile running at a 2 GHz clock frequency, the maximum interconnect wire delay for all three shapes ranges from 67% to 78% of one clock cycle. For the 36 mm$^2$ large processor tile running at 4 GHz, the maximum interconnect wire delay takes 4.0–4.7 clock cycles for the three shapes. The interconnect wires for both the medium sized and large processor tiles may be pipelined to increase throughput [111]. Compared with square-shaped tile for all sizes, the maximum wire delay of the house-shaped tile increases by 15.9% to 21.3% and the maximum wire delay of hexagonal-shaped tile increases by 2.8% to 8.7%.

For a fully-synchronous system, special design effort is required to balance the interconnect wire delay skew to increase the maximum achievable frequency. As shown in Table 5.2, the actual max link wire delay skew is smaller than the maximum link wire

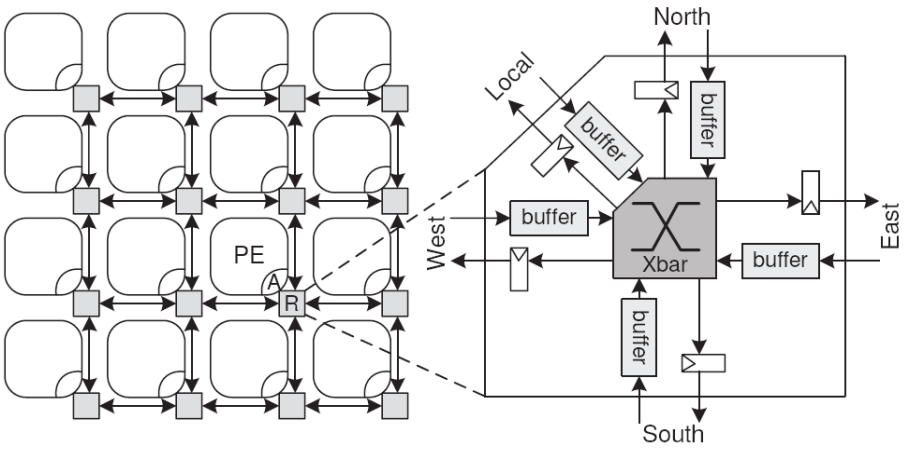



Figure 5.8: A 2D mesh processor array using five-port routers where one port connects to the local processor core.

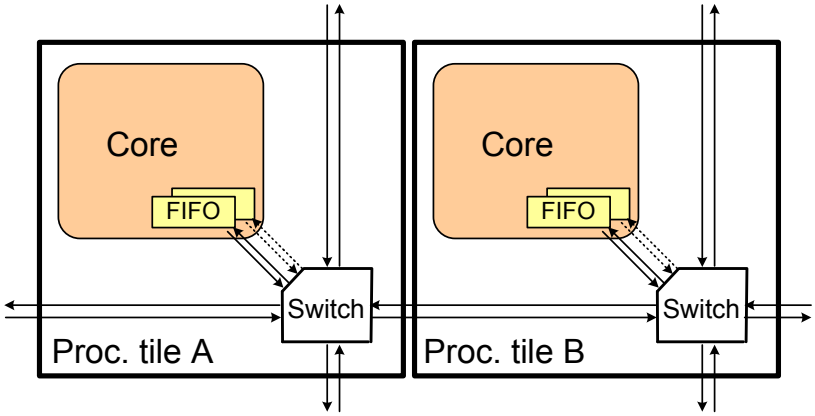



Figure 5.9: A diagram of two processor tiles in the *4-4 Rect* mesh processor array with four interconnection links and two input ports per tile.

delay. For the 0.04 mm$^2$ small processor tile and the 4 mm$^2$ medium sized processor tile running at a 2 GHz clock frequency, the maximum interconnect wire delay skew for all processor shapes takes around 5% and 55% of one clock cycle, respectively. For the 36 mm$^2$ processor tile running at 4 GHz, the maximum interconnect wire delay skew is around 3.7 clock cycles on average. Compared with square-shaped tiles for all sizes, the maximum wire delay skew of the house-shaped tile increases by 8.7% to 29.7% and the maximum wire delay of hexagonal-shaped tile increases by 20.1% to 49.1%. The results suggest the placement of switch fabrics for non-rectangular tiles has higher impact on link wire delay skew than the square tile.

## 5.4 Application mapping

### 5.4.1 Target Interconnect Architecture

The proposed topologies can be used for dense on-chip network with either dynamic routers or static circuit switches. Figure 5.8 shows the inter-processor communication in a typical 2D mesh processor array using dynamic routers. As the diagram shows, processors are connected by 5-port routers each with five buffers and one 5 by 5 crossbar. The dynamic routers also include hardware logic to implement different routing algorithms for different topologies.

The static circuit-switch interconnection has smaller area, lower power dissipation and lower complexity than dynamic router interconnection. In this work, we assume processor tiles are connected with circuit switches which are suitable for applications with steady communication patterns. Figure 5.9 shows the *4-4 Rect* mesh array using circuit switches each with four nearest-neighbor interconnection links and two ports connecting to the processor core. In this case, each processor is capable of taking two inputs from the four directions and sending data to all four directions. The long distance communication is performed by software in the intermediate processors. The circuitry diagram of other topologies is similar, which differs in the number of links among neighboring processor tiles.

### 5.4.2 Two Benchmark Applications

Parallel programming on the discussed many-core systems with dense on-chip networks includes two steps: 1) partitioning the algorithms at a fine-grained level; 2) mapping the tasks to the nodes of the processor array and connecting the nodes with available links defined by the topology [112]. In order to compare all discussed topologies, two complete applications including an H.264/AVC residual encoder and an 802.11a/g OFDM baseband receiver are manually partitioned and mapped onto all topologies.

Figure 5.10 shows a 22-node task graph of an H.264/AVC residual baseline encoder

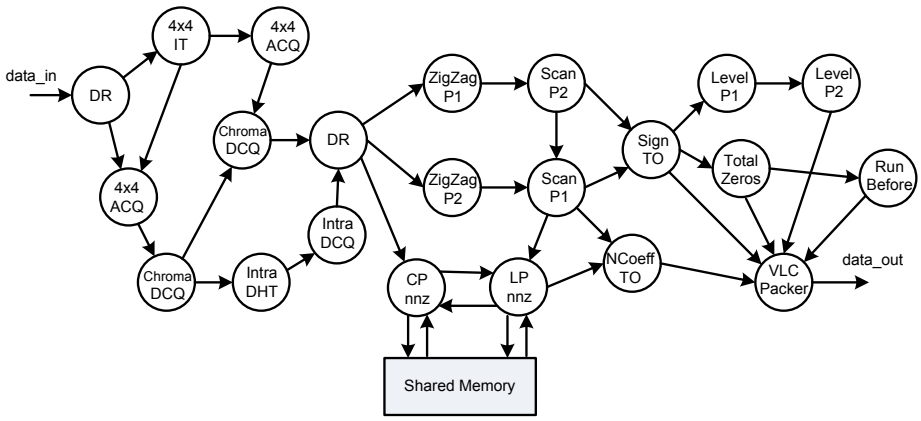



Figure 5.10: Task graph of a 22-node H.264/AVC video residual encoder.

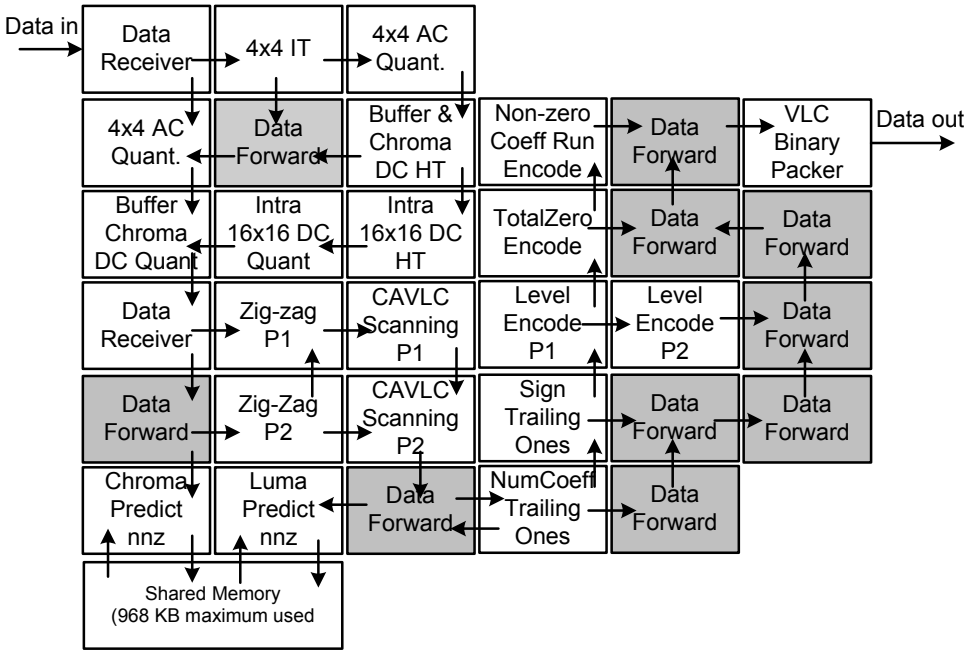



Figure 5.11: An H.264/AVC video residual encoder mapped on a processor array with *4-4 Rect* mesh topology. The processors in gray are used for merging and forwarding data.

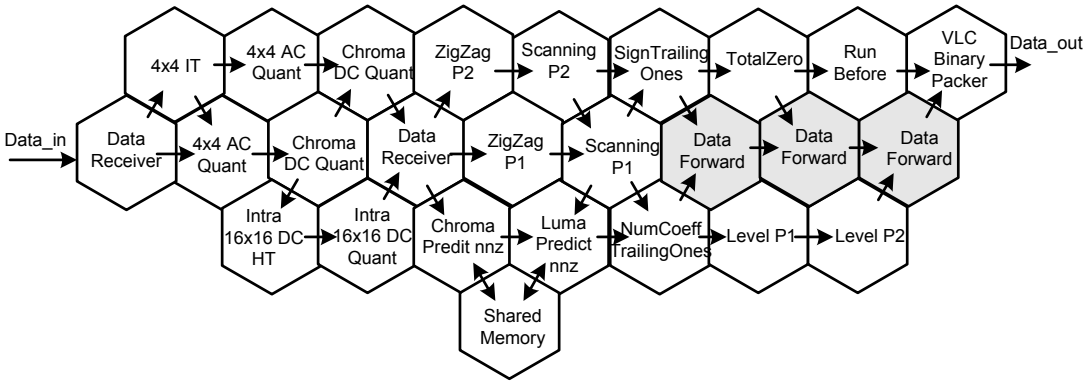



Figure 5.12: An H.264/AVC residual video encoder mapped on a processor array with *6-6 Hex* topology.

composed of integer transform, quantization and context-adaptive and variable length coding (CAVLC) functions [112]. The encoder also requires a shared memory module as shown in the task graph.

Figure 5.11 shows an example mapping of the H.264/AVC residual encoder capable of 1080p HDTV encoding at 30 fps (frames per second) on the baseline *4-4 Rect* mesh that uses 32 processors plus one shared memory. The *4-4 Rect* mesh is inefficient in handling a complex application like H.264/AVC encoding. A total of 10 processors are used for merging and forwarding data which accounts for 31% of the total number of processors.

Figure 5.12 shows a possible 25-processor mapping on the proposed *6-6 Hex* topology. The hexagonal-shaped processors accept a maximum of two inputs from the six nearest-neighbor processors. Compared with the design using a *4-4 Rect* mesh, seven processors are saved, which accounts for a 22% reduction in the total number of processors.

Figure 5.13 shows a 22-node task graph of a complete 802.11a/g WLAN baseband receiver which is computation-intensive requiring two dedicated hardware accelerators: Viterbi decoder and FFT. The complete receiver includes necessary practical features such as frame detection, timing synchronization, carrier frequency offset (CFO) estimation and compensation, and channel estimation and equalization [113]. Figure 5.14 shows a mapping of the 802.11a/g baseband receiver (54 Mbps) on the baseline *4-4 Rect* mesh that uses 32 processors plus the Viterbi decoder and FFT accelerators with 10 processors used for

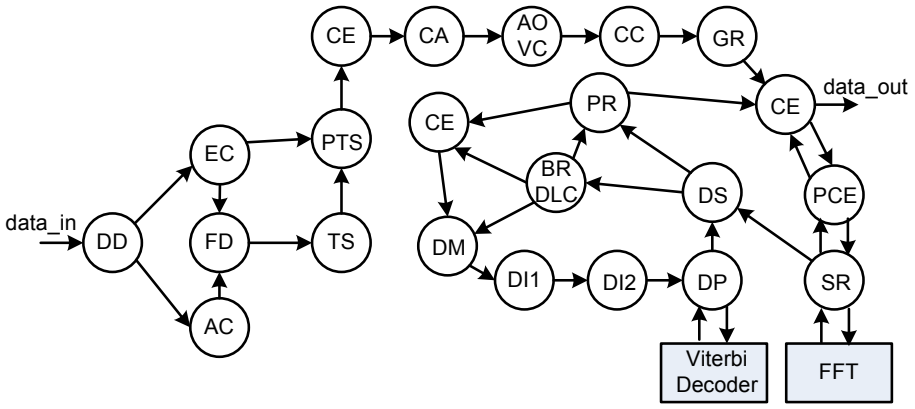



Figure 5.13: Task graph of a 22-node 802.11a/g WLAN baseband receiver

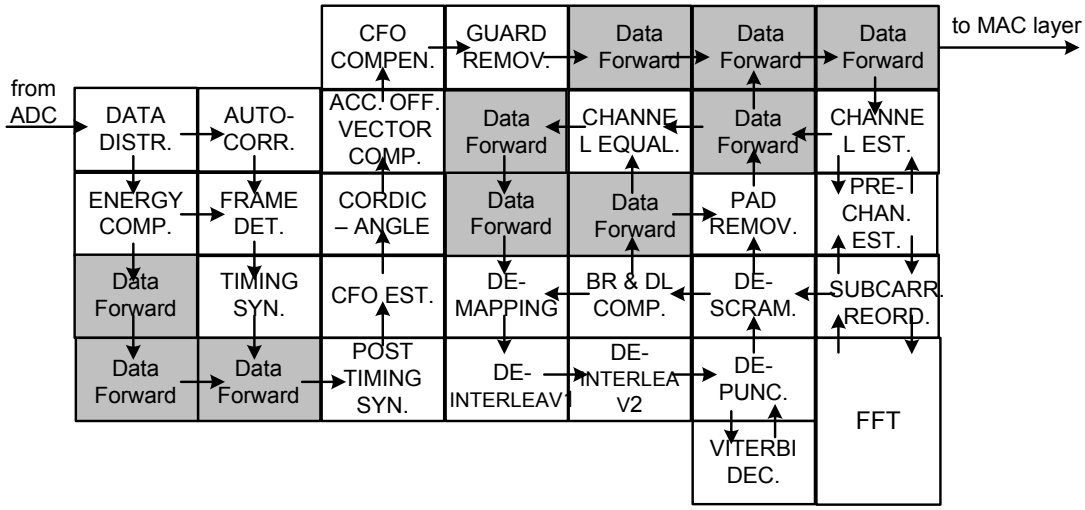



Figure 5.14: An 802.11a/g baseband receiver mapped on the processor array with baseline *4-4 Rect* mesh topology.

merging and forwarding data.

Figure 5.15 shows a mapping on the hexagonal-shaped tile architecture which requires only 24 processors plus the Viterbi decoder and FFT accelerators—25% fewer processors than those used in the *4-4 Rect* mesh mapping.

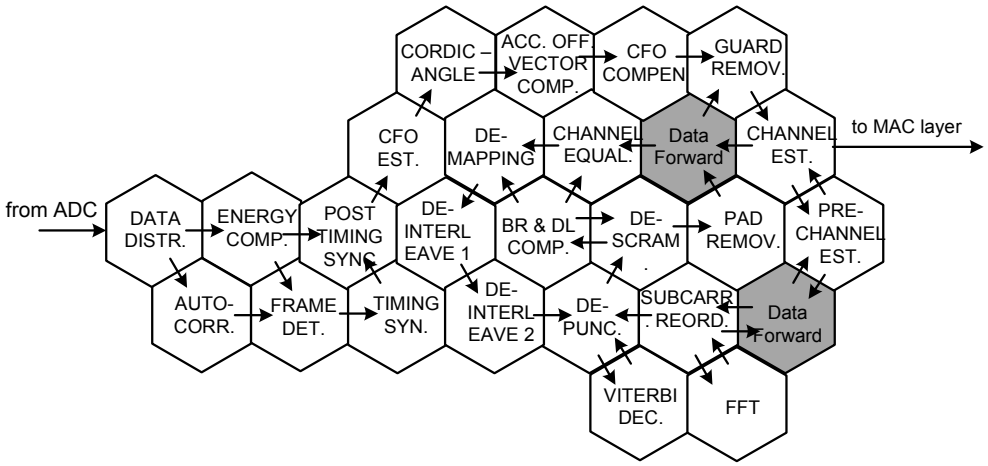



Figure 5.15: An 802.11a/g baseband receiver mapped on the processor array with *6-6 Hex* topology.

### 5.4.3 Application Mapping Results

**Total number of used processors**

All six proposed topologies (type (b)–(g) in Figure 5.2) are much more efficient than the *4-4 Rect* mesh (type (a) in Figure 5.2), resulting in processor count reductions of 16% to 22% for the H.264 encoder and 19% to 25% for the 802.11a/g baseband receiver as shown in Figure 5.16. The results are the same for the *5-5 House* and *5-5 Rect Alt. Offset* architecture due to essentially the same topology property. Similarly, the *6-6 Hex* has the same result as the *6-6 Rect Offset* architecture. The number of used processors of the *8-8 Rect* and *8-4 Rect* meshes is smaller than the *5-5 House* and *5-5 Rect Alt. Offset* topologies because of more communication links between processors. However, the two 8-neighbor Rect meshes require a slightly larger number of processors than the two 6-neighbor topologies which yield the largest processor number reduction (24%) compared to the *4-4 Rect* mesh. This is because the communication patterns of the two applications are mostly localized. Thus, topologies with more nearest-neighbor links yield more benefits than topologies with fewer nearest-neighbor links.

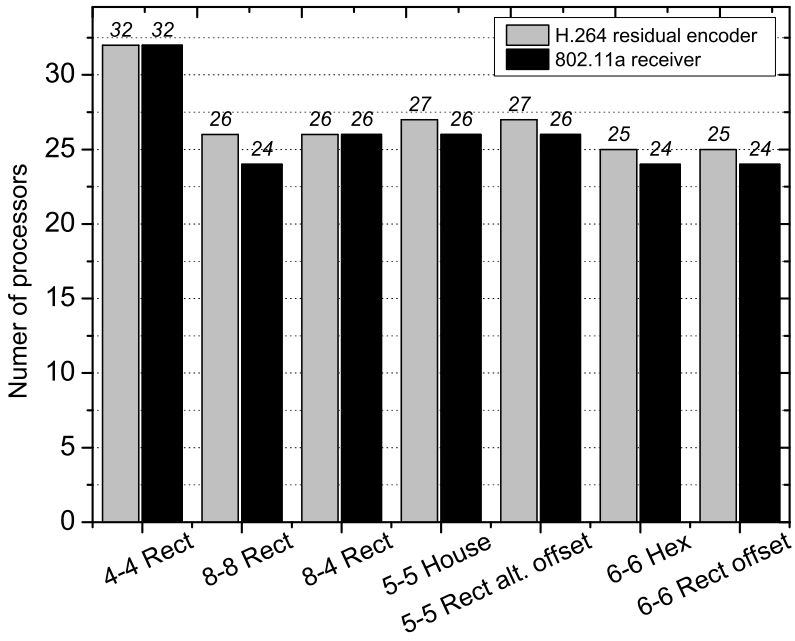



Figure 5.16: The number of processors used for mapping two applications to the seven topologies (type (a)–(g) in Figure 5.2).

**Total communication link length**

The total communication link length for the two applications can be calculated based on either Euclidean or Manhattan link length as shown in Table 5.1 and the application mapping diagrams.

Figure 5.17 shows the total communication length based on non-Manhattan-style wires. The *8-8 Rect* and *8-4 Rect* have an average of 3% and 9% longer communication lengths than the *4-4 Rect* mesh because they use more long communication links. The *6-6 Hex* and *6-6 Rect Offset* are the most efficient topologies, yielding the largest reduction (19%) in average total communication link length compared to the baseline *4-4 Rect* mesh.

Figure 5.18 shows the total communication link length based on Manhattan-style wires. All proposed topologies result in a slight increase of the total link length ranging from 1% to 5% compared to the baseline *4-4 Rect* mesh. This small link length increase because of Manhattan wires has little influence in application performance, area and power consumption, which will be demonstrated by the following physical implementation results.

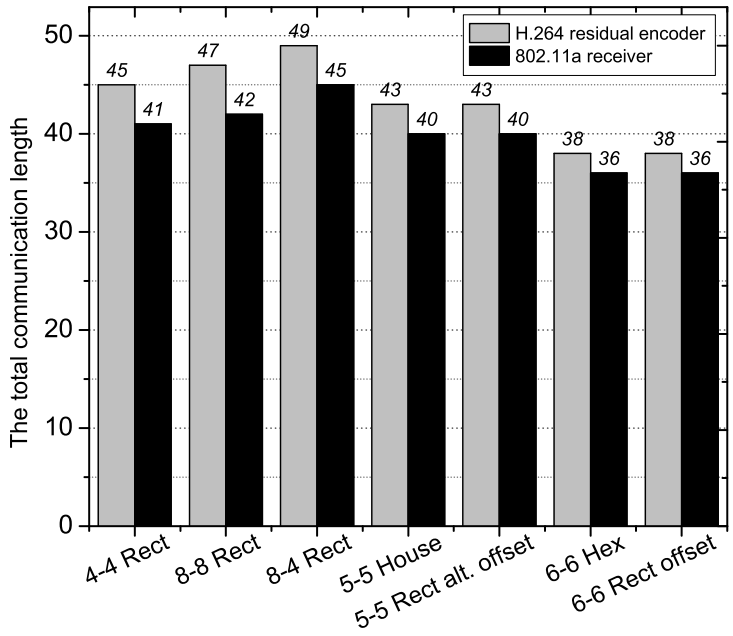



Figure 5.17: The total communication length based on non-Manhattan-style wires (Euclidean link length) for the two applications mapped on the seven topologies (type (a)-(g) in Figure 5.2). The link length is estimated based on the assumption that the area of each processor tile is equal to one square unit of the length.

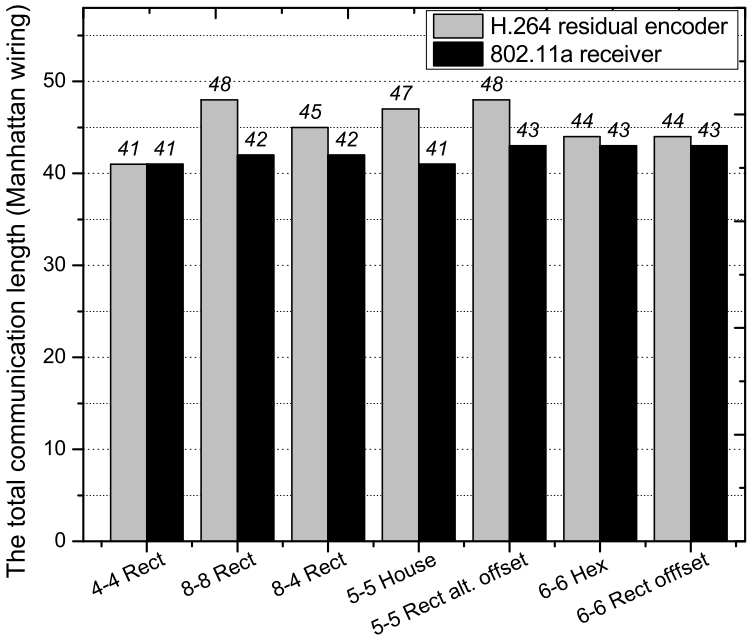



Figure 5.18: The estimated total communication length based on Manhattan-style wires for the two applications mapped on the seven topologies (type (a)–(g) in Figure 5.2). The link length is estimated based on the assumption that the area of each processor tile is equal to one square unit of the length.

## 5.5 Non-rectangular Processor Tile Physical Design

This section presents the design methodology of implementing a fully functional processor and corresponding CMPs based on the proposed seven topologies.

### 5.5.1 Physical Design Methodology

For performance evaluation, a small processor with configurable circuit-switch interconnection is used for all physical designs. The processor contains a 16-bit datapath with a 40-bit accumulator and 560-Byte instruction and 256-Byte data memories. Each processor also contains a configurable clock oscillator and two 128-Byte FIFOs for data buffering and synchronization between two processors [73, 74]. Each inter-processor link is composed of 19 signals including a clock, 16-bit data and 2 flow-control signals [93]. This processor is tailored for all topologies under test with a different number of neighboring interconnections ranging from 4 to 8. The internal switch fabrics are changed accordingly. The hardware overhead is minimal for 5-neighbor, 6-neighbor and 8-neighbor processors with only 0.5%, 0.7% and 2.0% hardware overhead based on synthesis results. The two 8-neighbor topologies add more complexity because processors communicate with two far-away processors via dedicated links as shown in Figure 5.2. In order to make CMP integration simpler, four additional sets of pins are inserted into the processor netlist after synthesis and are directly connected with bypass wires. This adds routing congestion in the corner for the topology shown in Figure 5.2(b) and across the processor tile for the topology in Figure 5.2(c).

All processors are implemented with a fully automated design flow spanning from RTL description to layout-level verification with STMicroelectronics 65nm CMOS technology. The processors are synthesized from Verilog with Synopsys Design Compiler and laid out with an automatic timing-driven physical design flow with Cadence SoC Encounter. Timing is optimized after each step of the physical design flow: floorplan, power planning, cell

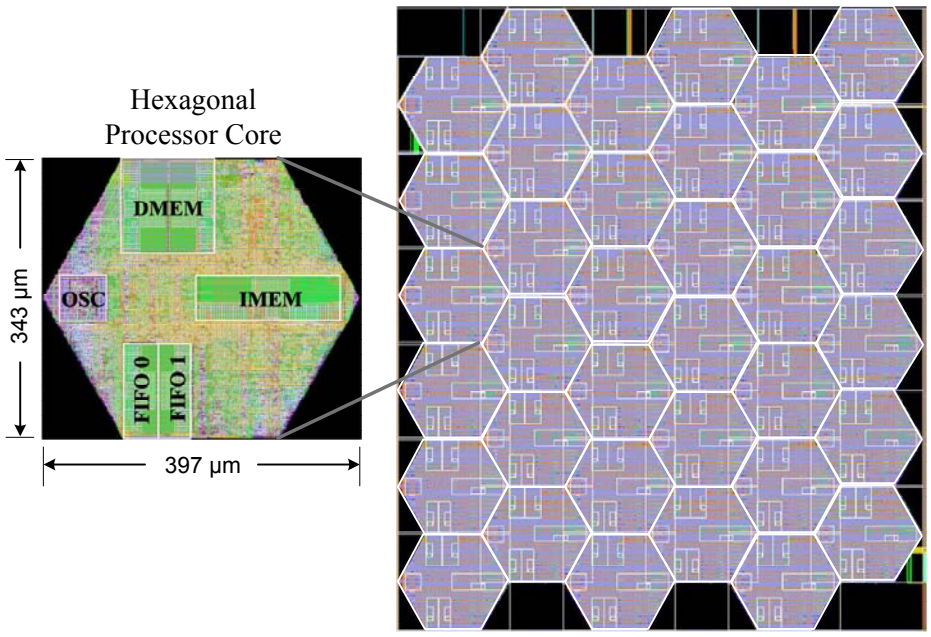



Figure 5.19: DRC clean and LVS clean layout of a hexagonal processor and a 6 by 6 multiprocessor array.

placement, clock tree insertion and detailed routing. A configurable oscillator (OSC) is manually designed from standard cells and laid out separately.

### 5.5.2 Non-rectangular Processor Tile and CMP Design

The house-shaped tile and hexagonal-shaped tile bring challenges for physical implementation. The first challenge to design the hexagonal processor is how to create a hexagonal shape at the floorplan stage. Rectangular placement and routing blockage in SoC Encounter are used to create approximate triangle corner blockages with each rectangular blockage differing by one unit in width and height. All rectangular blockages are piled together to create an approximate triangle in the four corners of the rectangular floorplan.

A proper placement of pin positions can help to achieve efficient global routing and easy CMP integration. At the floorplan stage, four sets of pins are put along the diagonal edge of the corner and two set of pins are placed in the horizontal top and bottom edge. Since all macroblocks have rectangular shapes (OSC, IMEM, DMEM and two FIFOs), this presents

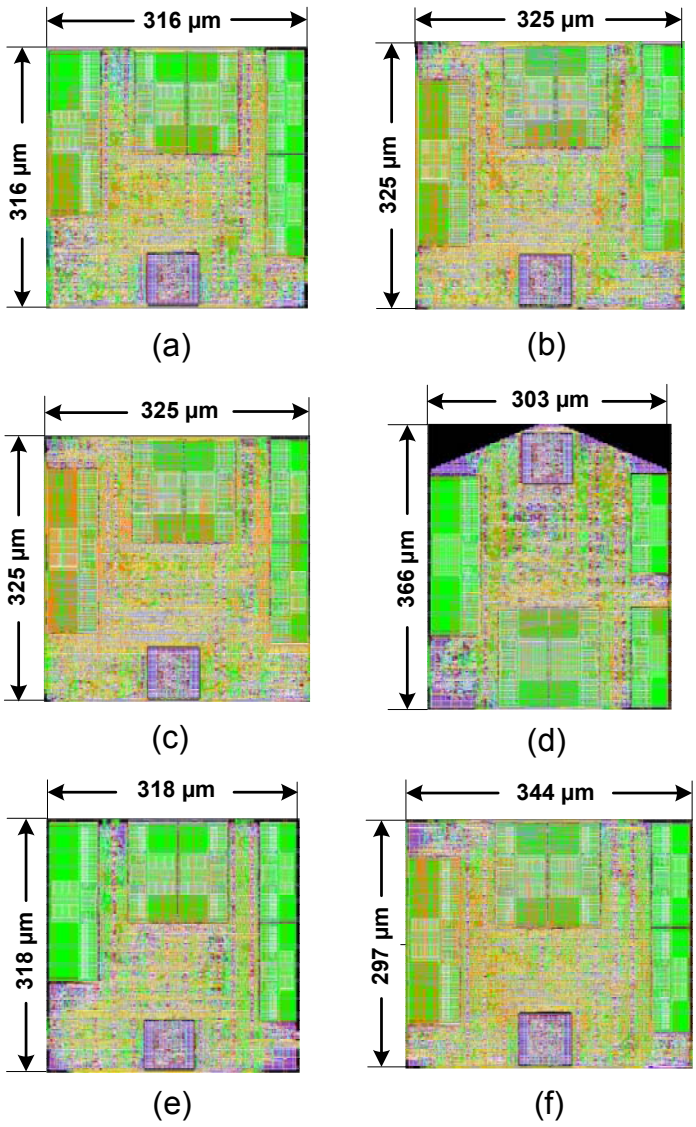



Figure 5.20: The final DRC and LVS clean processor tile layouts corresponding to topologies (a) 4-4 Rect, (b) 8-8 Rect, (c) 8-4 Rect, (d) 5-5 House, (e) 5-5 Rect Alt. Offset, and (f) 6-6 Rect Offset. The hexagonal tile (*6-6 Hex*) shown in Figure 5.19 is not included. All tiles have cell utilizations from 81% to 83%.

a challenge to place the macroblocks. In this design, the macroblocks are placed along the edge and the oscillator and IMEM are placed in the left and right corners, respectively as shown in Figure 5.19.

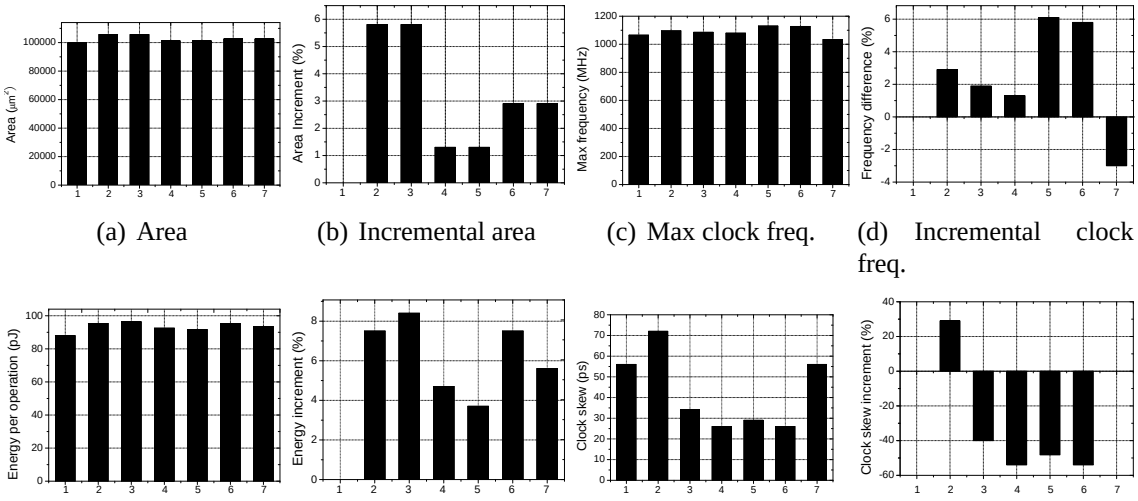
Metal 6 and metal 7 are used to distribute power over the chip and the automatically-created power stripes can stop at the created triangle edge in the corner. The power pins are created on the top and bottom horizontal edges. When integrating the hexagonal processor together, the power nets along the triangle edge can be connected automatically or manually by simple abutment.

Once a hexagonal processor tile is laid out, a script is used to generate the RTL files of the multiprocessor. The CMP array can be synthesized with empty processor tiles inside. Another script places the hexagonal tiles with the blockage area overlap with nearest-neighbor processors along the triangle edge of each hexagonal tile. SoC Encounter can connect all pins automatically although there are overlaps between LEF (library exchange format) files. The final GDSII files are read into Cadence icfb for design rule check (DRC).

Figure 5.19 shows the final layout of a hexagonal-shaped processor tile and a 6 by 6 hexagonal-tiled multiprocessor array.

## 5.6 Chip Implementation Results

All discussed topologies enable an easy integration of processors by abutment without global wires in the physical design phase. For all topologies, there is no long-distance inter-communication link across more than two processors and processors are pipelined in a way that the critical path is not in the interconnection links. Therefore, the maximum achievable frequency of an array is the same as an individual core, which is one of the key advantages of our proposed dense on-chip networks.




(a) Area (b) Incremental area (c) Max clock freq. (d) Incremental clock freq.

(e) Energy per operation (f) Incremental energy per operation (g) Clock skew (h) Incremental clock skew

Figure 5.21: Comparison of seven optimized processor tiles showing (a) absolute area; (b) incremental area relative to the *4-4 Rect* tile; (c) absolute maximum clock frequency; (d) incremental clock frequency relative to the *4-4 Rect* tile; (e) absolute energy per operation; (f) incremental energy per operation relative to the *4-4 Rect* tile; (g) absolute clock skew; (h) incremental clock skew relative to the *4-4 Rect* tile. The processor types 1 to 7 correspond to the topologies shown in Figure 5.2 (a) to (g) which are *4-4 Rect*, *8-8 Rect*, *8-4 Rect*, *5-5 House*, *5-5 Rect Alt. Offset*, *6-6 Hex* and *6-6 Rect Offset*.

### 5.6.1 Processor Tile Implementation Results

Seven tile types are implemented from RTL to GDSII layout to get reliable estimates of how the topologies affect the system performance in nanoscale chip design. All floorplans use the same power distribution design and the I/O pins and macroblocks are placed along edges reasonably depending on the topology.

In standard-cell design, the cell utilization ratio has a strong impact on the implementation result. A higher cell utilization can both save area and increase system performance if the design is routable. In order to get a minimum chip area for all tiles, we start with a relatively large tile area which results in a small cell utilization ratio. Then the tiles are repeatedly laid out while maintaining the aspect ratio and reducing the area by 5% in each iteration with minor pin and macroblock position adjustments in the floorplaning phase. Once a minimum area within 5% has been reached, the area change is reduced to 2.5%. The layout tool is pushed until it is not able to generate an error-free GDSII layout for all tiles. Figure 5.20 shows the final layouts of the other six processor tiles besides the hexagonal tile shown in Figure 5.19. Our methodology results in high cell utilizations for all tiles ranging from 81% to 83%.

Figure 5.21(a) shows the absolute area of the seven processors and Figure 5.21 (b) shows the area increments compared to the baseline *4-4 Rect* tile which has the smallest area and the highest cell utilization of 83%. The hardware overhead of all processor tiles is very small. For the other six designs, the relative area increment is proportional to the number of nearest-neighbor connections. Compared with the baseline *4-4 Rect* tile, an area increase of 1.3%, 2.9% and 5.9% are required for the 5-neighbor, 6-neighbor and 8-neighbor tile designs, respectively. All six designs have a cell utilization of 81%.

Figure 5.21(c) depicts the maximum clock frequency of all seven designs and Figure 5.21 (d) shows the frequency increment relative to the baseline *4-4 Rect* tile which can operate at a maximum of 1065 MHz at 1.3 V. Due to an increase of area, the two 8-neighbor mesh tiles can operate at 1.9% and 2.9% higher frequency. The *5-5 Rect Alt. Offset* and *6-6*

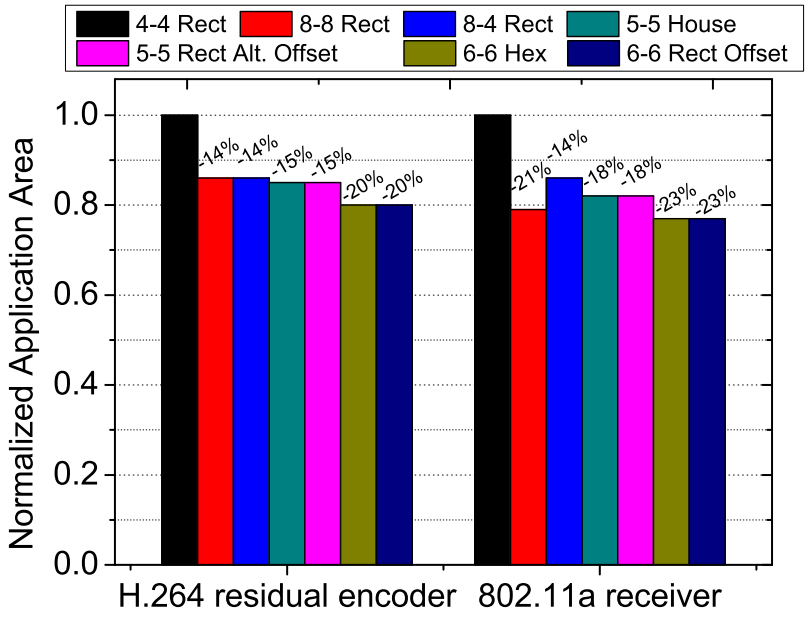
*Hex* tiles have noticeably higher frequencies than other designs which achieve a frequency increase of 6.1% and 5.8%, respectively. The *5-5 House* tile has the same processor logic design and area as the *5-5 Rect Alt. Offset* tile, while it has a frequency increment of only 1.5%. This is probably because the required aspect ratio for the house-shaped tile is not a good fit for this particular physical implementation. This can also explain why the *6-6 Rect Offset* tile has the lowest frequency, a reduction of 3.0% in maximum frequency compared to the baseline *4-4 Rect* tile.

Figure 5.21(e) shows the energy per operation and Figure 5.21 (f) shows the incremental energy per operation compared to the *4-4 Rect* tile. The energy is estimated based on a 20% activity factor for all internal nodes. All six proposed tiles have a higher energy per operation ranging from 3.7% to 8.4% because of the extra circuits for interconnections. Like the area increment, the average energy increments are proportional to the number of neighboring interconnections as shown in Figure 5.21 (f).

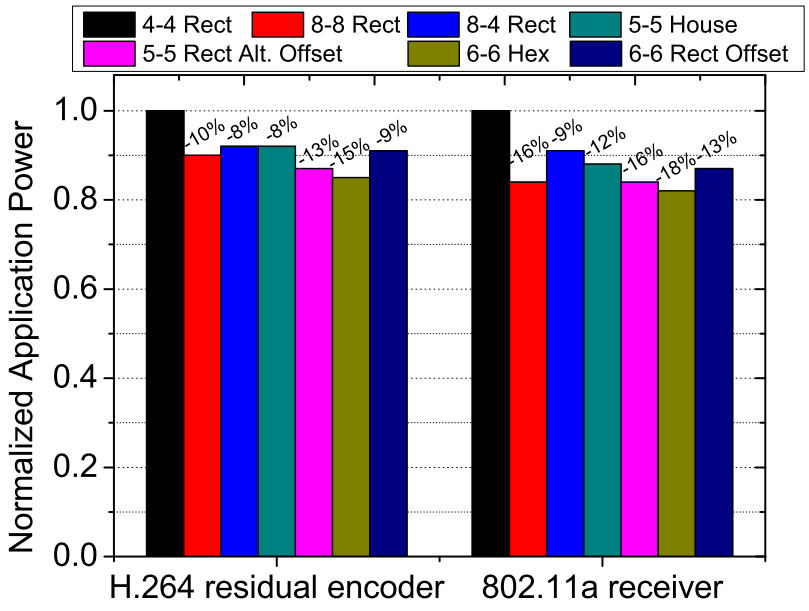
Figure 5.21(g) shows the worst-case clock skew for all seven processor tiles and Figure 5.21(h) shows the clock skew increments compared to the *4-4 Rect* tile. The *8-8 Rect* tile shows a 29% higher clock skew probably because routing congestion in the corners affects the clock tree synthesis. The more circle-like shape helps the layout tool to generate a clock tree with smaller clock skew. As expected, the house-shaped tile and hexagonal-shaped tile have the lowest clock skew with a reduction of 54% compared to the baseline *4-4 Rect* tile.

### 5.6.2 Application Area

Application area depends solely on the number of used processors and the processor tile sizes if processors are compactly tiled. Figure 5.22 shows the normalized application area of two benchmark applications for all seven topologies. Compared with *4-4 Rect*, the six proposed topologies reduce application area by 14% to 22%. Corresponding to the largest reduction of the number of used processors, *6-6 Hex* and *6-6 Rect Offset* achieves




(a)




(b)

Figure 5.22: The final mapping results of the H.264 residual encoder capable of HD 1080p encoding at 30 fps and 802.11a/g baseband receiver in 54 Non-rectangular Processor Tiles Design and CMP IntegrationMbps mode (a) normalized application area, and (b) normalized power consumption.

the largest application area savings, a 22% reduction compared to the *4-4 Rect*.

### 5.6.3 Application Power

For applications mapped to the many-core processor array, the average power can be estimated by:

$$P_{Total} = \sum_i P_{Core,i} + \sum_i P_{Comm,i} + P_{other} \quad (5.1)$$

where $P_{Core,i}$ and $P_{Comm,i}$ represent the power consumption of processor core and communication circuits of the $i^{th}$ processor. $P_{other}$ is the average power of other chip components such as memory modules or accelerators. The power consumption of processor core can be estimated as:

$$P_{Core,i} = \alpha_i \cdot P_{CoreActive} + (1 - \alpha_i) \cdot P_{CoreStall} \quad (5.2)$$

where $\alpha_i$ is the processor activity factor; $P_{CoreActive}$ and $P_{CoreStall}$ are the average processor power consumption while the processor core is 100% actively executing instructions and stalling (executing no-ops). Simulation results show that $P_{CoreRunning} \approx 2 * P_{CoreStall}$ for all topologies. The two applications are simulated based on the *4-4 Rect* topology to collect the computational processor activity factors and their output link activity factors. Due to a minimal workload change on computational processors across different topologies, the computational processor activity factors of all topologies are almost the same. Simulation results show that the average computational processor activity factors are 58% for H.264/AVC residual encoder and 49% for 802.11a/g baseband receiver, respectively. The activity factors of routing processors are estimated based on the number of input links and the corresponding link activity factors. The routing processor activity factors are 9.0% and 18.2% for H.264/AVC residual encoder and 802.11a/g baseband receiver, respectively.

The communication power of processor *i* can be estimated as follows:

$$P_{Comm,i} = \sum_{j} (\delta_{ij} \cdot P_{CommActive,L_j} + P_{CommIdle,L_j}) \quad (5.3)$$

where $\delta_{ij}$ is the communication active percentage of link *j*;
$P_{CommActive,L_j}$ and $P_{CommIdle,L_j}$ are the average power consumed by a link with a length *L* while the link is 100% active and idle. The communication link power is estimated based on simulation which is in a range of 5% to 10% of the processor power consumption. The link idle power (mainly leakage power) is nearly zero due to the simplicity of the communication circuits.

The voltage and frequency scaling are considered for more accurate power estimation. In order to meet the throughput requirement for the two mapped applications, processors need to run at 959 MHz at a supply voltage of 1.15 V for H.264 residual encoder and 594 MHz at a supply voltage of 0.92 V for 802.11a/g baseband receiver. All processors run at the same clock frequency and supply voltages.

Based on the above equations as well as the processor power consumption numbers, application mapping diagrams, the required clock frequencies and supply voltages for processors, Figure 5.22(b) shows the normalized average power consumption of the H.264 residual encoder (encoding 1080p video at 30 fps) and the 802.11a/g baseband receiver (54 Mbps) for all seven topologies.

Compared to *4-4 Rect*, the six proposed topologies reduce application power by 9% to 17%. The *6-6 Hex* achieves the largest average application power savings, a 17% reduction compared to *4-4 Rect*. The *5-5 Rect Alt*. Offset is the second most power-efficient topology, yielding 15% average power consumption compared to *4-4 Rect*. Although the *6-6 Rect Offset* has essentially the same topology property as *6-6 Hex*, it reduces only 11% application power compared to *4-4 Rect*.

## 5.7 Conclusion

This chapter presents seven low area overhead and low design complexity topologies other than the commonly-used 2D mesh for dense on-chip networks. The proposed topologies include two 8-neighbor meshes, two 5-nearest-neighbor and three 6-nearest-neighbor topologies—three of which use a novel house-shaped and hexagonal-shaped tile. Two complete applications are mapped onto all topologies for realistic comparisons. Commonly available commercial CAD tools are used to implement tiled CMPs for all proposed topologies including the two non-rectangular processor tiles. The application mapping and chip implementation results demonstrate the effectiveness of the inter-processor interconnect of all proposed topologies. Compared with 2D mesh, the hexagonal-shaped 6-nearest-neighbor topology reduces 22% application area and 17% average power consumption with a 2.9% area increase per processor tile. The rectangular-shaped 6-nearest-neighbor topology provides the same interconnect architecture as the hexagonal-shaped tile. Despite being less power-efficient, its simpler physical design makes it an attractive design alternative for many-core dense on-chip networks.

# Chapter 6

# Efficient Distributed On-Chip Shared Memory for Video Applications

The memory wall problem has long existed due to the fact that the bandwidth and latency of main memory have not kept pace with CPU performance [114, 115]. This speed disparity has widened significantly for many-core systems which have limited memory pins and hundreds or even thousands of memory-hungry cores. The primary solution to the memory gap has been the implementation of multi-level memory cache hierarchies. However, the cache consistency and coherency problems emerge for many-core systems, which require significant hardware or software overhead to enable data sharing between multiple processors.

For multimedia applications, typically the workload has regular memory access patterns and small memory requirements, which makes alternative architectures attractive. The goal of this research is to explore the design of a distributed shared on-chip memory system for fine-grained many-core architecture where each processor operates independently and asynchronously.

Table 6.1: An estimate of memory requirements for DSP and video algorithms

| **Applications** | **Memory Size (Bytes)** |
|---|---|
| 64-point FFT | 256 |
| 1024-point FFT | 4096 |
| 8x8 DCT | 64 |
| Motion Estimation (48 x 48 search range) | 2304 |
| 16x16 Intra Prediction | 416 |
| 1080p CAVLC | 1684 |
| 1080p deblocking filter | 456 |
| Adaptive Loop Filter (HEVC) | 1562 |
| Sample Adaptive Offset (HEVC) | 640 |

## 6.1 Background

### 6.1.1 Video Application Memory Requirements

The memory requirements of DSP and multimedia algorithms depend on the coding approach and the amount of parallelism exploited. A theoretical lower bound of data memory is the minimum memory size required by an algorithm to work efficiently without data swapping. For most algorithms, this lower bound can be easily determined. For example, a double-buffered 1024-point complex FFT requires approximately 4096 words of memory storage [116]. Table 6.1 lists an estimate of memory requirements of several DSP and video algorithms including two filter algorithms for the next generation high efficiency video coding standard (HEVC). The estimation is based on either C implementations such as Motion Estimation, Deblock Filter and the two HEVC filter algorithms, or assembly implementations on the current AsAP system such as FFT, 8x8 DCT, 16x16 Intra Prediction and 1080p CAVLC. The memory requirements of these tasks range from several hundred bytes to several KBs. Some of the algorithms require additional line buffer for system-level integration such as deblock filter, adaptive loop filter and sample adaptive offset. The line buffer sizes are in the range of several KBs to a dozen KBs.

### 6.1.2 Current AsAP Memory System

#### Motivation of Small Memory Processors

Modern processors typically spend a significant percentage of die area for memories which might occupy over half of the chip area [117]. Large memories reduce the area available for computation units, consume significant power, and require longer memory access latencies. Based on the observation of small memory requirement of DSP and multimedia tasks, the first generation AsAP processor uses 26% area for memories per core with 64-word 32-bit instruction and 128-word 16-bit data memory [117]. The second generation of AsAP chip spends 18% die area on memories per core with 128-word 35-bit instruction memory and 128-word 16-bit data memory [18]. In order to support applications requiring large memories, the current AsAP system also includes three 16-KB on-chip shared memories supporting connections with up to four programmable processors, and each contains a single-port SRAM that can range up to 64 KWords or 128 KB [118]. The shared memory can reach a peak throughput of one read or write per cycle. In addition, each port supports least-recently-serviced priority arbitration during times of simultaneous access by multiple processors. In order to integrate the memories into the GALS array, each port contains an input and output FIFO, and the block contains a local clock oscillator.

#### Memory Limitations for Mapping Video Applications

The current AsAP memory system is efficient for mapping: 1) DSP and multimedia kernels, and 2) applications with high task-level parallelism and small local memory requirement. However, there exist some limitations: 1) there is no capacity hierarchy and a large capacity gap between the local 128-word data memory and 16-KB shared memory, 2) the latency to access the large shared memory is high due to the cost of synchronization FIFOs, 3) the large shared memory is accessible only from processors that are adjacent to the memory, which results in a waste of top or bottom processors used as memory controllers,

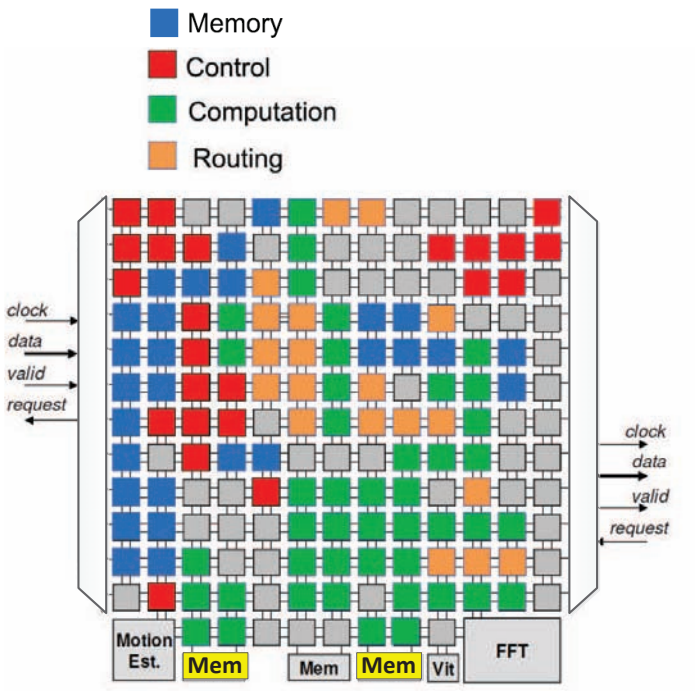



Figure 6.1: A full H.264 baseline encoder mapped to AsAP platform [119].

and 4) the area efficiency of the shared memory is low with only 37% area spent on SRAM itself.

The current memory system is not efficient in handling video applications requiring larger local memories due to data dependencies. We have mapped a full H.264/AVC encoder to the second generation AsAP system as shown in Figure 6.1. Three memory intensive tasks in the H.264 encoding are the current/reference frame management, motion vector management and non-zero coefficient management in entropy encoding. They arise from the fact that the encoding is based not only on the current macro-block but also on previously encoded macro-blocks. The encoder uses 115 AsAP processors, two shared memories and the motion estimation accelerator [120]. A total of 33 processors are used solely for storing temporary data, which incurs high area and power overhead.

In order to alleviate the memory system limitation, we have proposed a bufferless shared memory modules to bridge the gap between the large buffered shared memory mod-

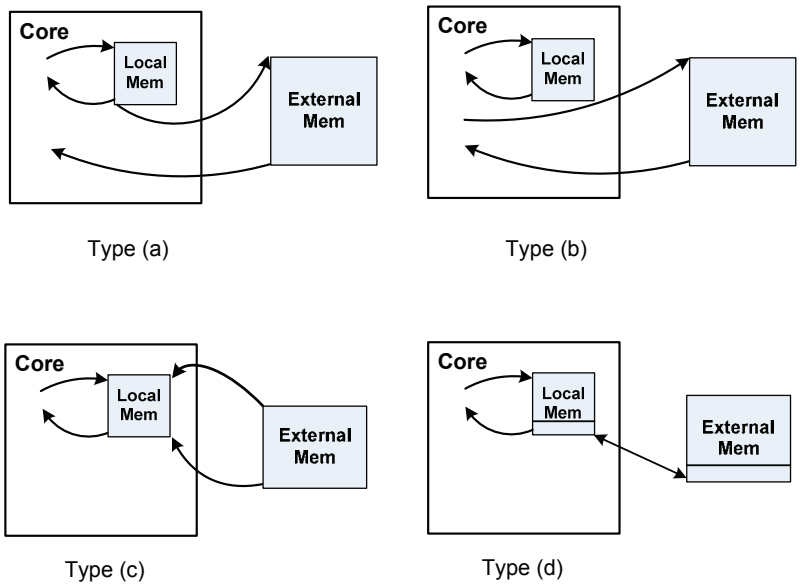



Figure 6.2: The four basic data memory organizations: (a) single address space, (b) separate address space, (c) cache, and (d) software-managed memory.

ules and small local memory. The novel source synchronous bufferless shared memory can enable safe memory sharing across different clock domains with low access latency and high throughput.

## 6.2 Shared Memory Primary Architecture

There are a variety of design options for adding a larger amount of data memory to a GALS many-core architecture like AsAP. This section first discusses the available primary architectures which determine the relationship between processor's local memory and on-chip shared memory and the programmer's view of the shared memory system.

### 6.2.1 Single Processor's View

Figure 6.2 shows four basic data memory organizations from the viewpoint of one processor. These organizations differ in the relationship between local memories (LM) and on-chip external memories (EM). The processor in type (a) has a single address space for

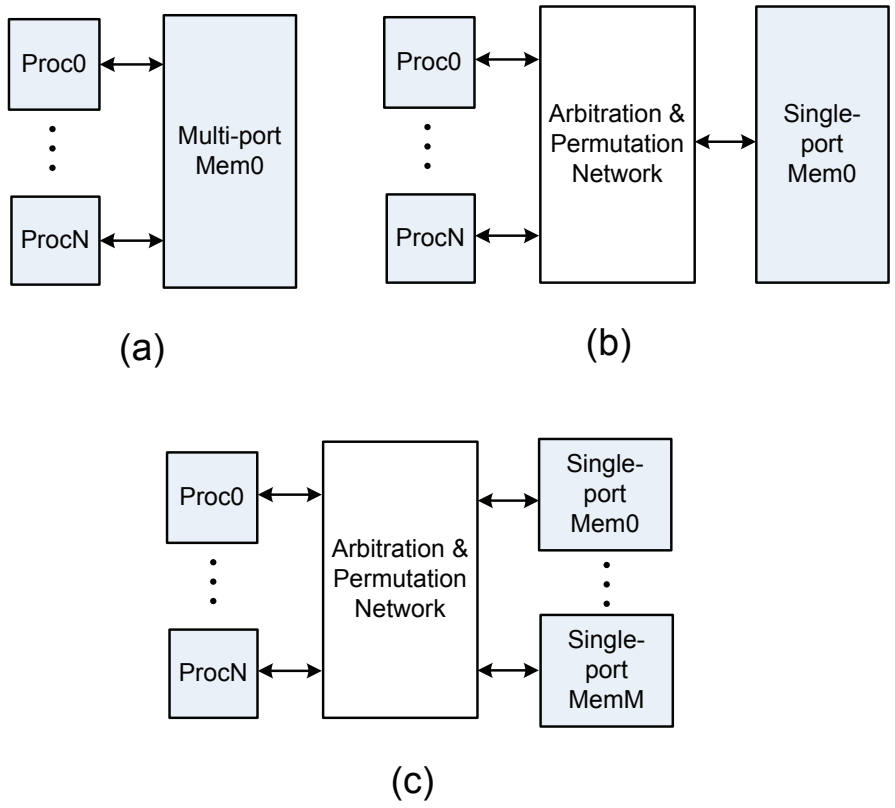



Figure 6.3: Three basic organizations of on-chip shared memory systems with (a) one multi-port memory, (b) one single-port memory and (c) multiple single-port memories.

LM and EM. EM can be considered as an extension of the memory space to the LM. Type (b) has separate address spaces for LM and EM. The two memories independently interact with the processor core. Type (c) is a traditional cache architecture where LM contains a subset of data in the EM. Thus, there are duplicate data in the memory system. In type (d) organization, LM and EM have separate address space. However, processors only operate on data from LM and software manages the movement of data from EM to LM. Type (d) is like a software-management cache without duplicate data in the system.

Type (d) also has the merit that the design of the shared memory module has very little impact on processing elements. The shared memories in previous generation AsAP use type (d) organization which is efficient for handling streaming applications like video encoding. This work continues to use this organization for the proposed bufferless shared memory module.

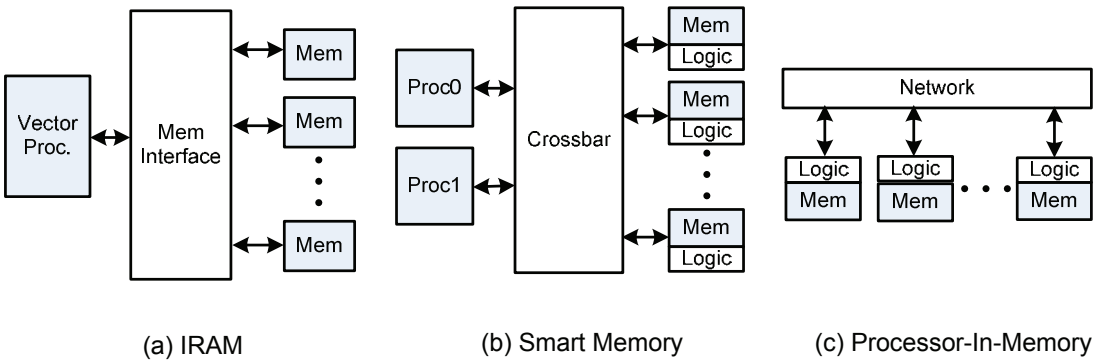



Figure 6.4: Three related on-chip memory systems: (a) Intelligent RAM (IRAM) [121], (b) Stanford Smart Memory [122], and (c) Processor-in-Memory (PIM) architecture such as FlexRAM [123] and the distributed PIM for motion estimation [124]

### 6.2.2 Sharing Among Multiple Processors

Depending on the memory organizations, Figure 6.3 shows three types of shared memory architecture. Type (a) allows processors to access the memory simultaneously through independent memory ports. Processors in type (b) share a single-port memory through an arbitration and permutation network. Processors in type (c) share a group of single-port memories through an arbitration and permutation network. Multi-port memories are expensive in terms of area compared with single-port memories. The previous generation AsAP uses type (b) architecture where four processors share one single-port memory.

### 6.2.3 Related and Proposed Memory Architecture

Figure 6.4 shows three related on-chip memory systems. Figure 6.4(a) shows the Intelligent RAM (IRAM) architecture which integrates a vector processor with wide datapath and multiple DRAMs onto a single die [121]. The IRAM has one memory address space and memories are not shared among processors.

Figure 6.4(b) shows the Stanford Smart Memory system which is a modular reconfigurable architecture targeting at reconfigurable computing applications [122, 125]. Smart Memories are composed of SRAM cells and configurable fabrics, which allows on-chip

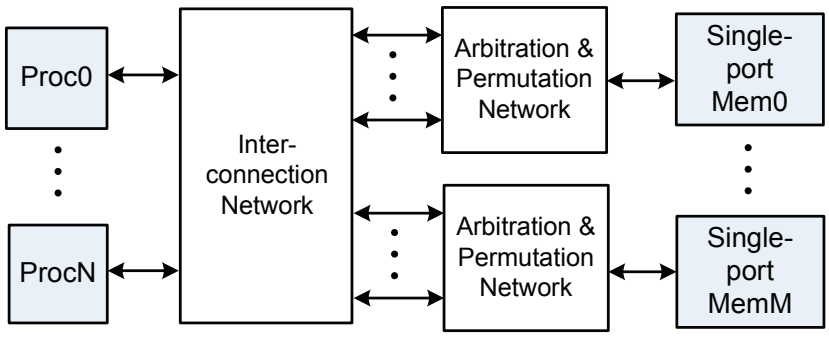



Figure 6.5: The proposed shared on-chip memory system allowing multiple processors to access multiple single-port memories via an interconnection network and an arbitration and permutation network.

SRAM resources to be configured as caches, buffers, or scratch-pad memories. Smart Memories can be configured to have a unified address space for both processors or separate address spaces for each processor. The flexibility of the Smart Memories system results in a 32% area overhead, and 23% power overhead for a 16-KB SRAM capacity [126] at 0.18-$\mu$m CMOS.

Figure 6.4(c) shows the processor-in-memory (PIM) architecture where hardware logic and memories are tightly integrated into a single tile which communicates with each other through an on-chip network. There is no direct memory module sharing and each processor utilizes separate memory address spaces. Examples of processor-in-memory architecture are the FlexRAM [123] where the hardware logic is a general-purpose processor, and the distributed PIM for motion estimation where the hardware logic is motion estimation ALUs [124].

Figure 6.5 shows the proposed on-chip memory architecture where multiple processors are capable of accessing multiple single-port memories via the interconnection network. Each single-port memory supports a small number of input requests with the help of the arbitration and permutation network. Each memory module may have different address space for one processor or they can be configured to share a unified address space for one processor.

Table 6.2: Area, access time and power consumption of various-sized SRAMs at 65 nm CMOS technology and 1.1 V supply voltage

| **Sizes (KB)** | **Area ($mm^2$)** | % of AsAP **Core Area** | **Access Time (ns)** | **Read Dynamic Power (mW)** | **Leakage (mW)** |
|---|---|---|---|---|---|
| 1 | 0.015 | 9.0% | 0.439 | 13.1 | 0.8 |
| 2 | 0.022 | 12.8% | 0.479 | 11.0 | 1.5 |
| 4 | 0.037 | 21.9% | 0.515 | 17.2 | 3.0 |
| 8 | 0.088 | 51.9% | 0.545 | 20.4 | 5.9 |
| 16 | 0.147 | 86.4% | 0.617 | 25.5 | 11.4 |
| 32 | 0.280 | 164.4% | 0.702 | 40.7 | 20.8 |

## 6.3 Shared Memory Physical Parameters

The physical design parameters, such as memory capacity and density, ports and physical distribution, affect how the memory is integrated into a processor array.

### 6.3.1 Capacity

The capacity is the size of each single-port SRAM within a memory module. Table 6.2 lists the statistics of a single read/write port SRAM (area, access time and power consumption) of various-sized SRAMs at 65 nm CMOS technology and 1.1 V supply voltages. The data is estimated by CACTI memory model tool [127].

The area scales closely with the size of the SRAMs and the area of an 8 KB single-port SRAM is about half size of an AsAP processor core. As the SRAM capacity increases, the access delay and dynamic power per read operation also increases. The 8 KB single-port SRAM runs 24.1% slower and consumes 56% more power than the 1 KB SRAM. The leakage power also scales closely with the size of the SRAM. The 8 KB single-port SRAM consumes 6.4 times more leakage power than the 1 KB SRAM. The SRAM statistics suggest that a moderate size of shared memory module with moderate power and access time is suitable for the video applications which require up to several KB memories per task as shown in Table 6.1.

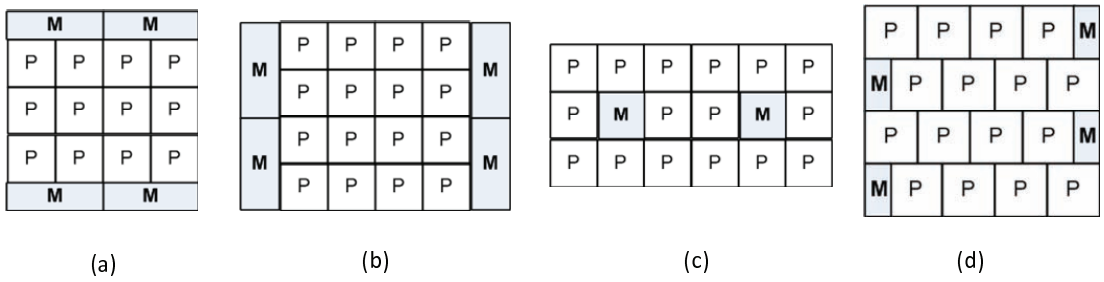



Figure 6.6: Various topologies for distribution of memories in an AsAP array: (a) memories at the top and bottom of the array, (b) memories at the left and right edge of the array, (c) processor tiles are replaced by memory tiles, and (d) memories at the left and right empty space of an array with *6-6 Rect Offset* topology.

### 6.3.2 Density

The density of the memory modules refers to the number of shared memory modules integrated into the AsAP system of a particular size. This parameter depends on available die area for memory, memory capacity and application requirements. As an example, the previous generation AsAP processor spends 18% area on local memories. A 16 KB shared memory module is twice the size of one AsAP processor. If the budget of the die area for memory is 25% of the total chip area, the 164-core system can integrate around eight 16 KB memory modules. With more memory modules, application data can be potentially partitioned among multiple memory modules to expose more parallelism.

### 6.3.3 Distribution

The distribution of memory modules within the array can take many forms and has a strong impact on application mapping. The memory modules can be placed at the four edges of the 2D mesh processor array as shown in Figure 6.6(a)(b). The processor tiles can be replaced by memory tiles as shown in Figure 6.6(c) where nearest-neighbor connections of some processors are lost. The memory modules can also be added to the empty space of certain topologies such as the 6-6 Rect Offset array as shown in Figure 6.6(d). The memory distribution can be a combination of the presented four forms depending on the area budget.

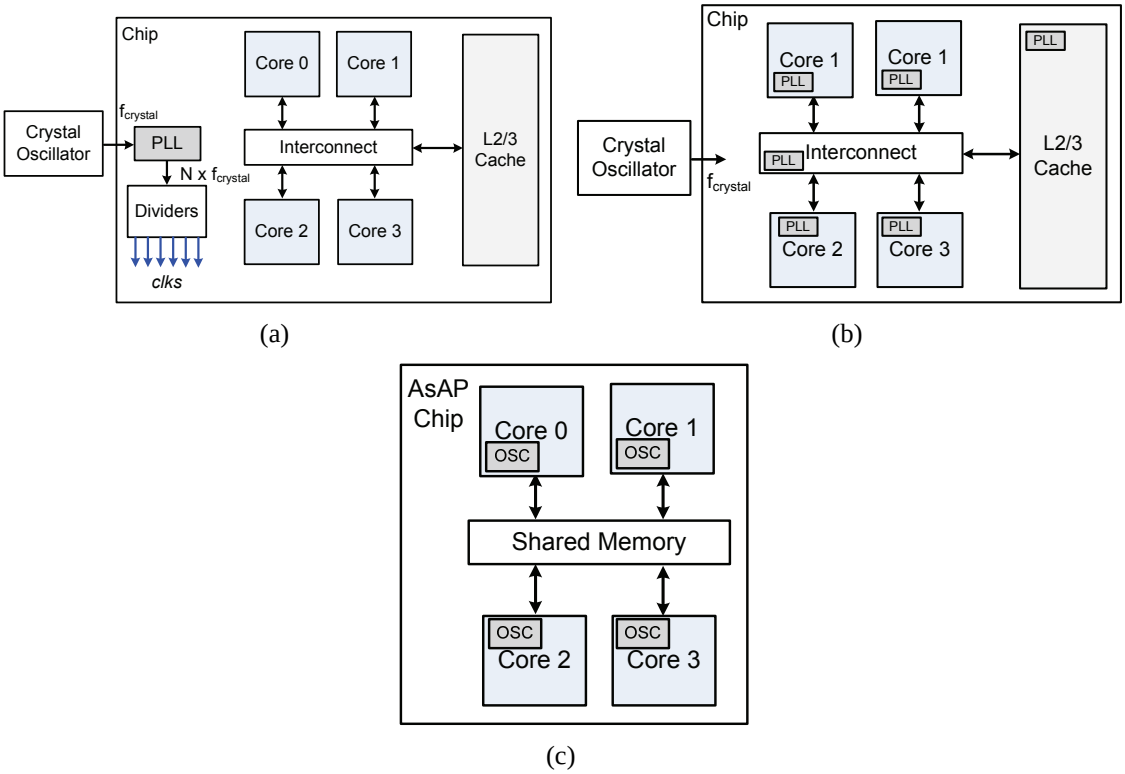



Figure 6.7: Shared memory clocking architectures use (a) related, (b) partially related, and (c) fully unrelated clocks.

## 6.4 Shared Memory Clocking Architecture

The clock architecture has the strongest impact on the shared memory module design. Figure 6.7(a)(b) shows two commonly used clock architectures for multi-core processors and their on-chip shared memory systems. Usually, four cores share a single L2/L3 cache and an external crystal oscillator is used to generate a low frequency reference clock. PLLs can be used to generate desired clocks for processors, interconnect and shared memories. All these chip components can operate either at related derived clocks from one PLL or at partially related clocks from multiple PLLs. A simple case of Figure 6.7(a) is that processors and memories use the same derived clock, which yields a fully synchronous system. A recent Intel 8-core Xeon processor adopts the latter clocking architecture, which contains 16 PLLs, 8 DLLs, and independent clock domains for each of the cores, caches, system interface and I/O regions [3].

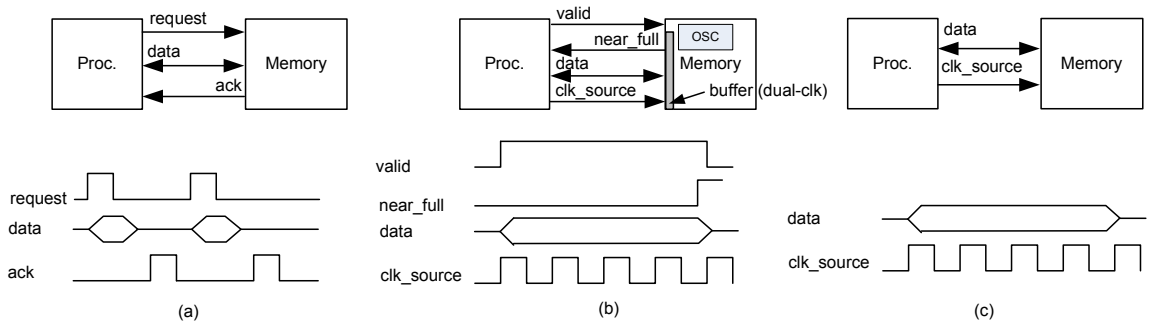



Figure 6.8: Three clocking source designs for the shared memory module on AsAP.

The AsAP processors operate at completely unrelated clock without an external reference clock as shown in Figure 6.7(c). Each core generates its own clock with a local ring oscillator [18]. Thus, the clock source for memory modules becomes a design parameter. In general, three distinct clocking strategies exist as shown in Figure 6.8.

First, the memory could be completely asynchronous, so that no clock would be required as shown in Figure 6.8(a). This solution severely limits the implementation of the memory module, as most SRAMs provided in standard cell libraries are synchronous.

Second, the memory can generate its own unique clock. The memory would be asynchronous to all processors in the array as shown in Figure 6.8(b). Dual-clock FIFOs are required to transfer data between shared memory modules and processors. The shared memory module in the second generation AsAP uses this clocking architecture [118]. However, FIFOs incur large area overhead and increase memory access latencies, which degrades performance for certain latency-sensitive applications.

Finally, a memory module can derive its clock from the clock of a AsAP processor as shown in Figure 6.8(c). The memory would then be synchronous with respect to this processor. In this case, dual-clock FIFOs are not required and memory access latency is much shorter than the second approach. However, this also brings the challenge to switch live clock when a memory module is shared by several processors with unrelated clocks.

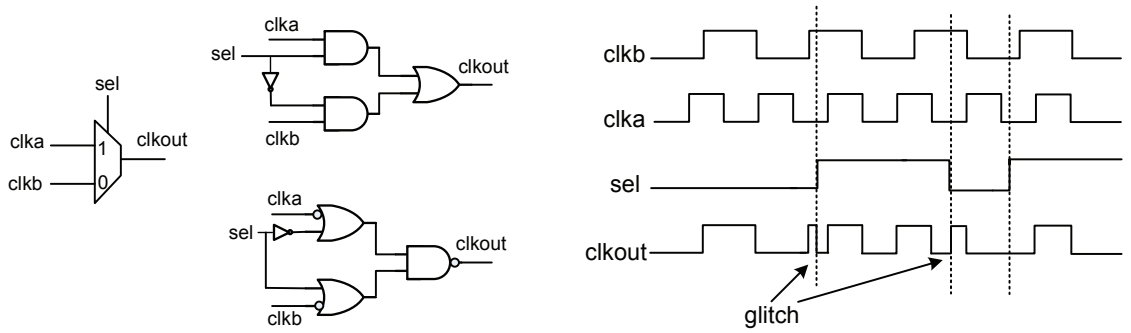



Figure 6.9: A simple clock switch multiplexer: (a) circuit, and (b) timing diagram.

## 6.5 Challenges and Solutions to Switch Live Clocks

Glitches on clock signals are hazardous to synchronous systems where all actions of circuits are coordinated by clocks. It is easy to generate hazardous glitches while switching the source of a clock line when a chip is running. Three main approaches to switch live clocks are discussed in the following subsections.

### 6.5.1 Approach 1: Simple Multiplexers

Figure 6.9 shows a simple implementation of a two-input clock switch, using either AND-OR or OR-AND type multiplexer logic. The control signal *sel* determines when to propagate *clka* or *clkb* to the output *clkout*. The problem with this simple switch is that the switch control signal can change at arbitrary time, thus creating a potential for chopping the clock at the output. Glitches can be generated due to an immediate switch from the current clock source to the other clock source. The timing diagram in Figure 6.9(b) shows how glitches are generated at the output *clkout*, when the *sel* control signal changes.

In order to be glitch free, the change of *sel* signal should be avoided at either clock's high state if two source clocks are unrelated. Furthermore, when the *sel* signal switches at both clock's low state, the output clock low state can be chopped which might create setup violations if data from the next clock arrives immediately at the first positive edge of the

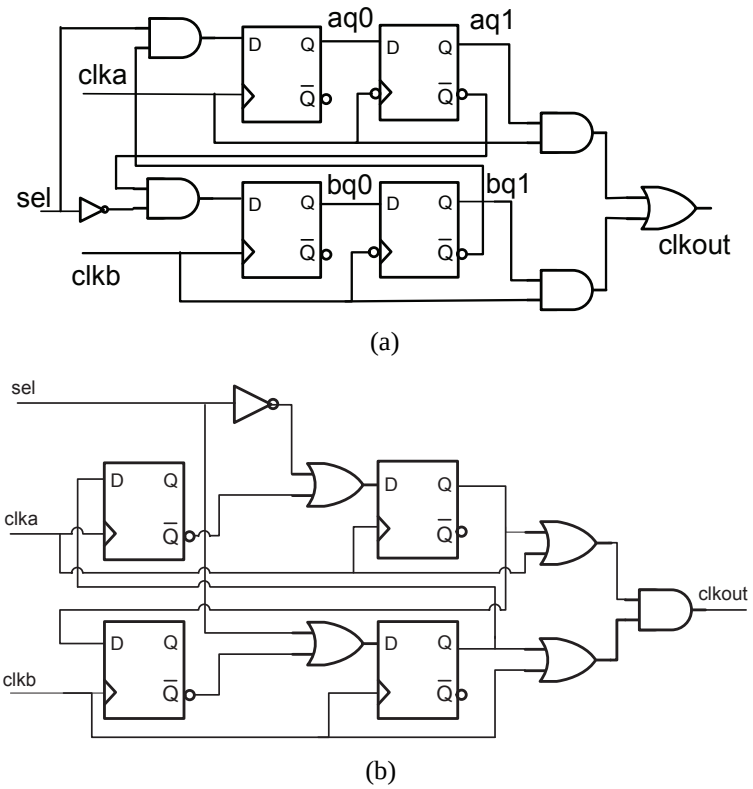



Figure 6.10: Two glitch-free clock switch circuits for unrelated clocks using (a) AND-logic circuit, and (b) OR-logic circuit.

output clock. This can be solved by extending the output clock low state or delaying the input data by at least one clock cycle.

### 6.5.2 Approach 2: Simple Multiplexers with Cross-coupled Synchronizers

Cross-coupled two-stage synchronizers can be added to the simple clock switch circuit to avoid glitches due to asynchronous select signals or feedbacks from one clock domain to the other. Figure 6.10 shows two popular glitch-free clock switch circuits [128]. Figure 6.11 shows an example of how the AND-logic clock switch circuit suppresses glitches during clock switches.

At the beginning, *sel* signal goes from low to high state when *clka* and *clkb* are both at high state. The outputs of the second-stage flip-flops *aq1* and *bq1* both will be at low state

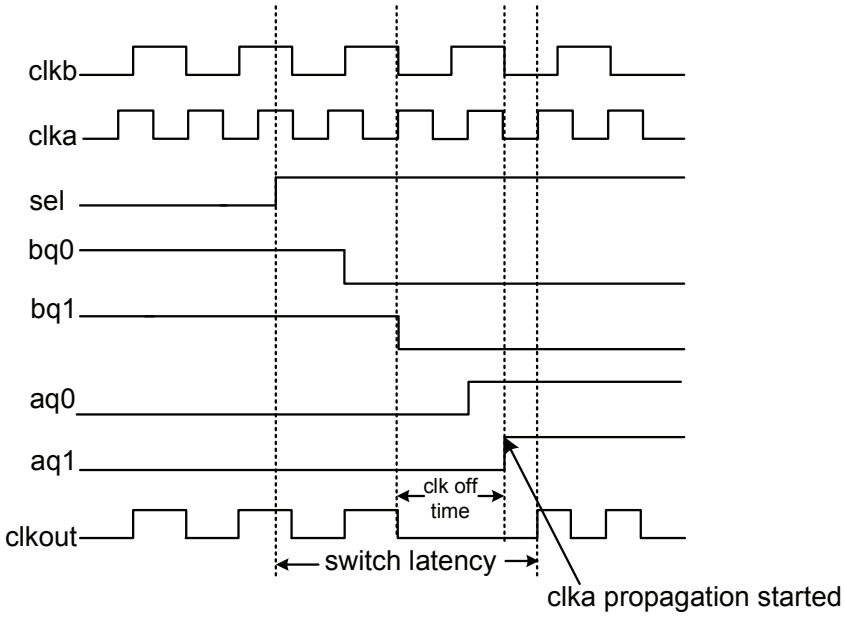



Figure 6.11: Timing diagram of the AND-logic glitch-free clock switch circuit.

for more than half a clock period of the next clock *clka*, which turns off the output clock. *clka* starts propagation when *aq1* turns high near the negative edge of *clka*. The circuit ensures that *clkout* can be at low state for more than one cycle before the first positive edge of arrives.

Assume that the clock periods of current clock source and next clock sources are $T_a$ and $T_b$, respectively. Depending on the switch timing of the *sel* signal, the clock switch time of the circuit falls into a range as shown in Eq. 6.1 and the worst-case switch time is $\frac{3T_a}{2} + 2T_b$. The switch latency is limited by the slowest input frequency of two switched clocks.

$$\frac{T_a}{2} + T_b \leq t_1 \leq \frac{3T_a}{2} + 2T_b \tag{6.1}$$

The OR-logic operates at only positive edge of input clocks as shown in Figure 6.10(b). The clock switch time of circuit in Figure 6.10(b) falls into a range as shown in Eq. 6.2. The worst-case switch time is $2T_a + 2T_b$.

$$T_a + T_b \leq t_2 \leq 2T_a + 2T_b \tag{6.2}$$

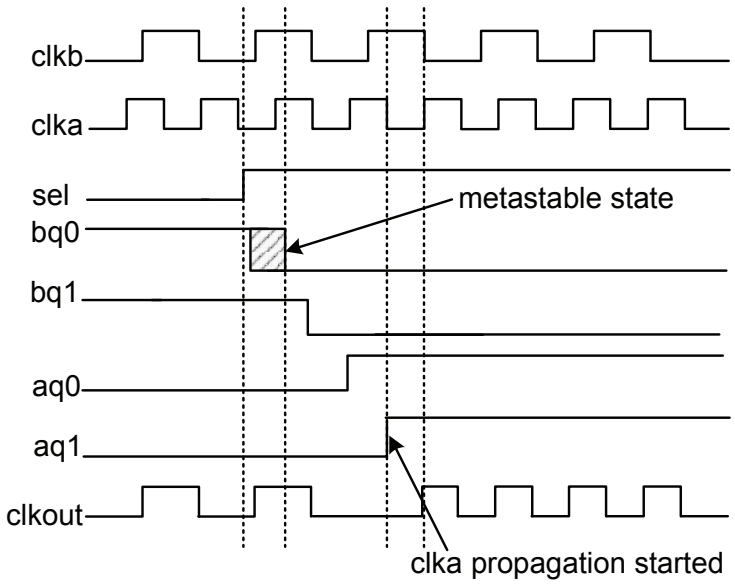



Figure 6.12: Metastability problem of the AND-logic clock switch circuit.

**Metastability Problem**

Metastability is a fundamental problem present when interfacing asynchronous blocks. Since *sel* is fully asynchronous to *clka* and *clkb*, the circuits in Figure 6.10 has potential metastability problems. Fig 6.12 illustrates the case where *sel* goes from low to high state near the positive edge of *clkb*, *bp0* could be metastable due to setup time violation.

A useful and prevalent approximation found in the literature for modeling the average failure rate due to metastability is shown in Eq. 6.3 [129].

$$\text{(Mean Time Between Failures) MTBF} = \frac{e^{t_r/\tau}}{T_0\ f_c\ f_i} \tag{6.3}$$

The variables in Eq. 6.3 are defined as follows. $f_c$ and $f_i$ are the sampling clock frequency and the input data event frequency. The parameter $\tau$ and $T_0$ are flip-flop parameters which are usually measured through experiments. $t_r$ is the resolution time (time since clock edge). The resolution time of *bq0* in Figure 6.10 is only half cycle of *clkb*, which severely decreases the MTBF when *clkb* operates at a high frequency.

For more reliable clock switch, additional synchronizers can be added. Figure 6.13

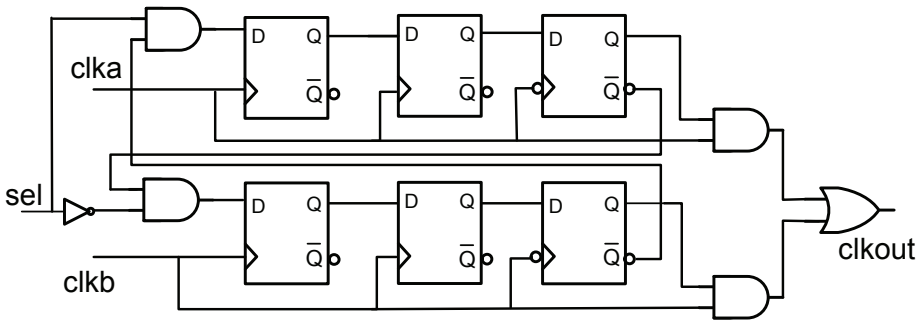



Figure 6.13: A 3-stage glitch-free clock switch circuit.

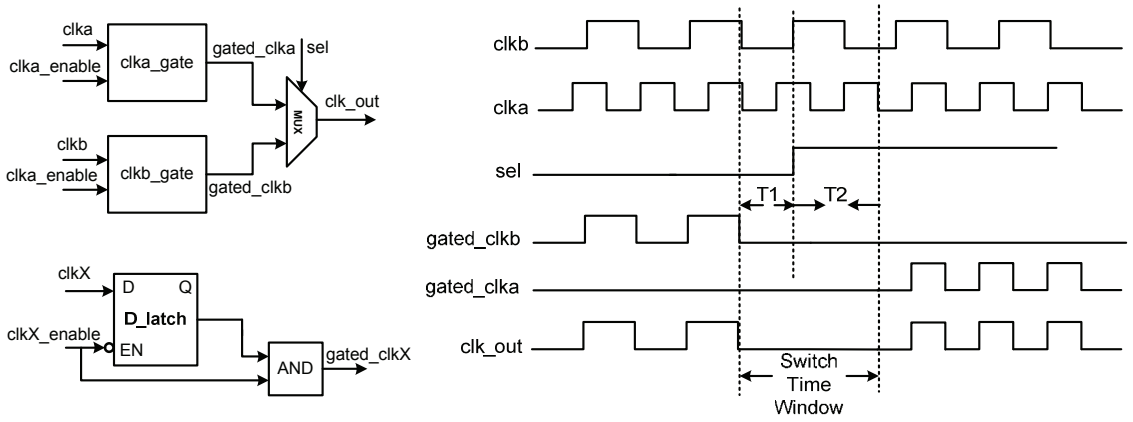



Figure 6.14: A simple clock switch circuit with clock gating to disable both clocks during clock switching (a) circuit, (b) timing diagram. For simplicity, the duplicate proc 1 circuits are not included.

shows a 3-stage AND-logic clock switch circuit. An additional flip-flop increases the resolution time by one more cycle. The circuit can be extended to have more synchronizers in between. However, the increased reliability comes at the cost of higher clock switch latency.

### 6.5.3 Approach 3: Simple Multiplexers with Clock Gating Circuits

Another solution to switch live clocks is to disable both input clocks during the transition of the *sel* signal. If the time is enough to disable one clock and enable the next, clocks can be cleanly switched by simple multiplexers without glitches.

Figure 6.14(a) shows a simple 2-input clock switch circuit with input clocks gated. A

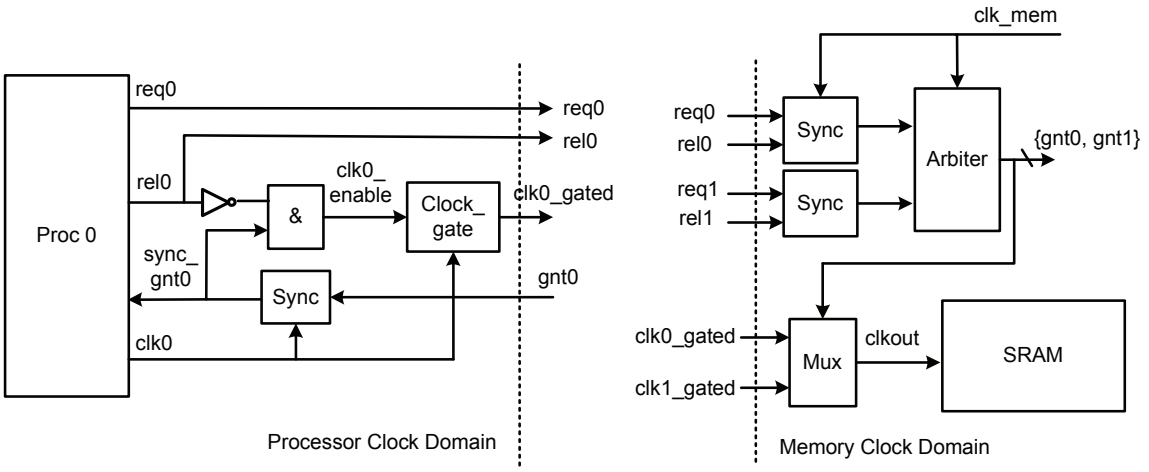



Figure 6.15: A block diagram of two processors sharing one memory module using a request-grant-release-ack protocol. Since Proc. 0 and Proc. 1 are the same, only Proc. 0 and the shared memory module are shown for simplicity.

clock gating circuit can be implemented by a low pass latch cascaded with an AND gate as shown in Figure 6.14(a). This circuit is often offered as a basic macro cell in many standard-cell libraries. The *clkX_enable* signal is latched at the low state of the clock. Figure 6.14(b) shows the timing diagram of the circuit switching *clkout* from *clkb* to *clka*. The circuit first disables *clkb* for a period of time T1 before the transition of signal *sel*. The enabling of *clkb* is delayed for a period of time T2 after the signal *sel* changes.

The clock enable signals and clock select signal need to be generated in a strict timing order. Fortunately, this timing can be guaranteed if processors use a request-grant-release-ack protocol to access a shared memory module.

Figure 6.15 shows an architecture of two processors sharing a memory module with clock gating and simple clock switch circuit. The two processors use unrelated clocks and the memory module also owns a local clock used for the operation of an arbiter. The request and release signals from processors are synchronized to the memory clock domain. The synchronizer comprises two or three flip-flops in series. The arbiter generates grant signals which are sent back to processors. The grant signals are also used as clock select signals for the clock switch multiplexer. At the processor side, the output clock is disabled

when the grant signal is low or the release signal is active high. This architecture uses the clock gating cell as shown in Figure 6.14(a).

Figure 6.16 shows the timing diagram of the proposed simple clock switch combined with the request-grant-release-ack protocol. At the beginning, the memory module is idle and *clkout* is disabled. Then processor 0 sends a *req0* signal to the memory module and the arbiter grants the access of processor 0 after three memory cycles. The *gnt0* signal is used to select gated *clk0* which is disabled until the synchronized *gnt0* signal arrives. The gated *clk0* starts propagation to *clkout* at the lower state of *clk0* when *gnt0* signal is asserted. The clock switch takes three memory cycles and two and a half *clk0* cycles since the *req0* is asserted. During this time, the request from processor 1 is not granted since processor 0 have not released the memory. Then processor 0 sends *rel0* signal which will disable *clk0* and *clkout* after half cycle of *clk0*. The release *rel0* signal results in the assertion of *grnt1* after three memory clock cycles. It takes another two and half cycle of *clk1* for *clk1* to propagate to *clkout*. The timing window when both clocks are off is larger than the sum of half *clk0* cycle and two and a half *clk1* cycle. It is enough for *clkout* to switch from *clk0* to *clk1* without glitches.

## 6.6 Processor-Memory Interconnection Network

The interconnection between processors and memories is another design parameter. AsAP processors communicate with each other using source synchronous circuit-switch network [117]. As shown in Figure 6.17, there are two types of processor-memory interconnection architectures. In Figure 6.17(a), memory modules can be attached to their nearest-neighbor processors. Non-nearest-neighbor processors need to go through processors attached to memory modules for memory access. The previous AsAP system uses this architecture which has drawbacks : (a) additional memory ports need to be added to the processor side, and (b) communication latency increases between non-nearest-neighbor

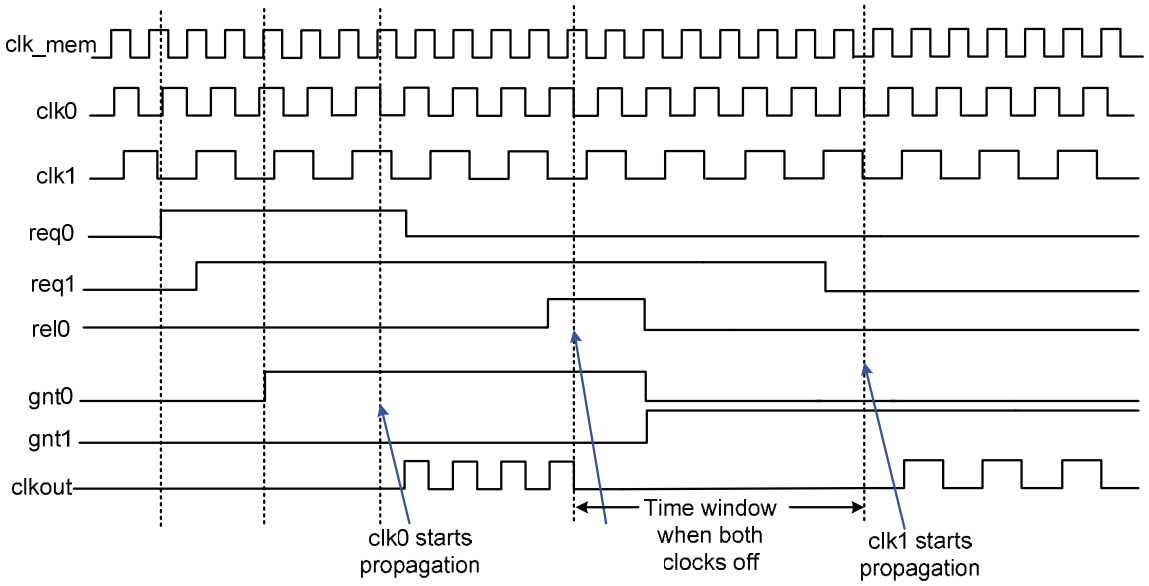



Figure 6.16: A timing diagram of the clock switch circuit using request-grant-release-ack protocol.

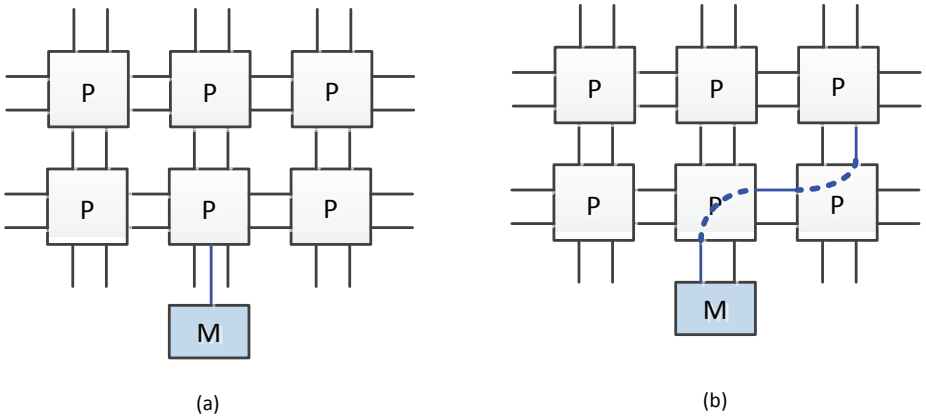



Figure 6.17: Two types of physical links to memories (a) nearest-neighbor interconnection with additional memory ports on the processor side, and (b) memory are treated as processor tiles and share the processor-processor interconnection links.

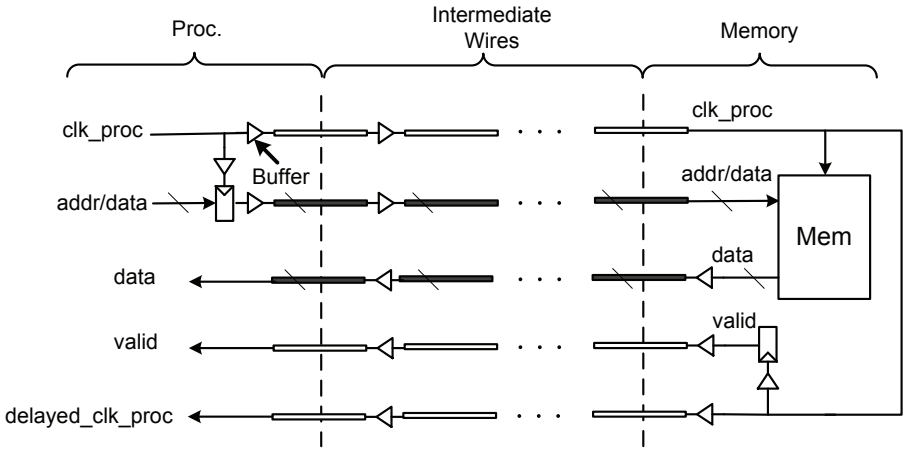



Figure 6.18: The physical links between processors and the bufferless memory module.

processors and memories. Figure 6.17(b) shows another design where memories are treated as processor tiles and processor-memory communication share the processor-processor interconnection links. Due to the fact that the interface signals of processor-memory interconnection are different from that of processor-processor interconnection, all the processor-processor interface signals need be adjusted to match the widths of the processor-memory interfaces.

Figure 6.18 shows a source synchronous processor-memory physical interconnection signal datapath. The processor sends *proc_clk* and *addr/data* to the bufferless memory module and the bufferless memory replies with *data*, *valid* and *delayed_proc_clk*. Depending on the distance between a processor and a memory module, the latency brought by those intermediate wires may take several clock cycles. This latency can be hidden by registering the data along the link [130]. However, due to the uncertain delays along the link, the phase difference between *clk_proc* and *delayed_clk_proc* is unknown. This may create timing violations as shown in Figure 6.19 if local *clk_proc* is used to sample the incoming memory data. In order to guarantee the good timing as shown in Figure 6.19(a), the following setup and hold time constraints need to be satisfied:

$$D + t_{clk-q} + t_{setup} < T \tag{6.4}$$

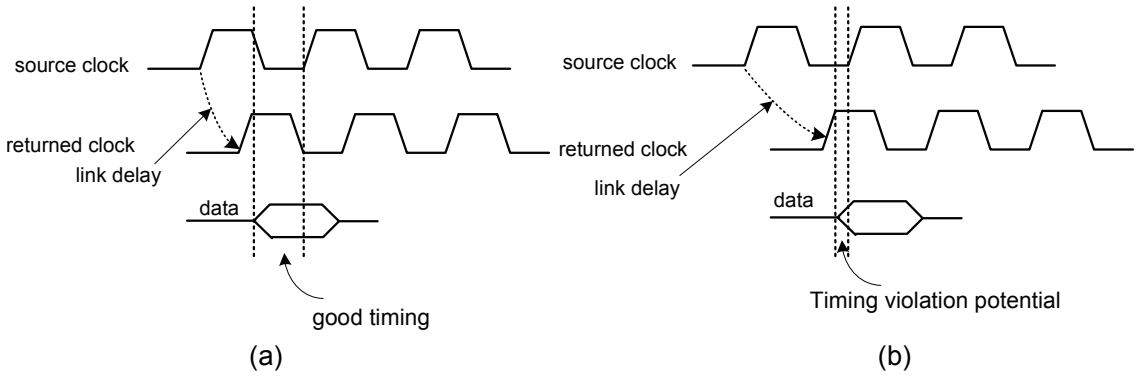



Figure 6.19: Timing diagrams of the processor-memory interface (a) correct timing, (b) timing violation.

$$t_{hold} < D + t_{clk-q} \tag{6.5}$$

Here, $D$ is the remainder of the total link wire delay divided by $T$. $t_{clk-q}$, $t_{setup}$ and $t_{hold}$ are the clock-to-output delay, setup time and hold time of a flip-flop, respectively and $T$ is the source clock cycle time.

Thus, a small dual-clock FIFO is required on the processor side to ensure reliable data transmitting. Since each AsAP processor core already has two input FIFOs and processor-processor and processor-memory interfaces are compatible, no additional FIFO needs to be added to the processor core. This reduces the memory port on the processor side, which is required by the FIFO buffered memory module design.

## 6.7 Bufferless Shared Memory Module

Meeuwsen et al. have proposed a four-port FIFO buffered shared memory module for AsAP [118] as shown in Figure 6.20. This architecture corresponds to the clock source type shown in Figure 6.8(b), where memory modules operate at their own independent clocks. This buffered memory module allows simultaneously requests from each processor and arbitrates at very fine-grained level between different requests. The FIFO buffered memory module achieves one word per cycle peak throughput for burst read and write mode. However, by incorporating both input and output dual-clock FIFOs, the buffered

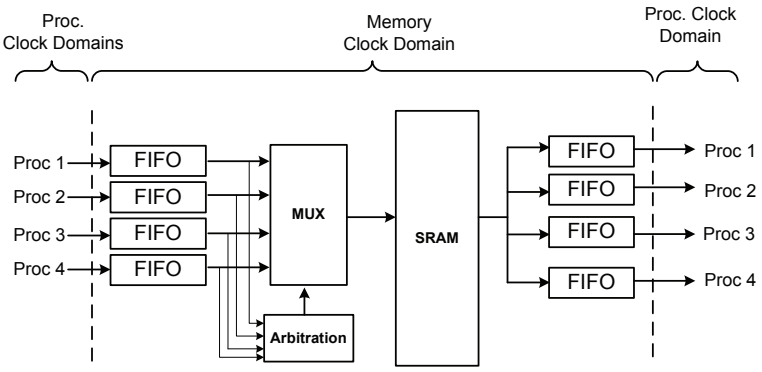



Figure 6.20: A four-port FIFO buffered shared memory module.

memory module has very low area efficiency (37% for SRAM cells) and very high access latency.

### 6.7.1 Primary Architecture

Most video encoding and decoding tasks have streaming feature that processors usually read or write a block of data at one time. Thus, memory modules do not need to serve the requests from processors at a fine-grained level. Instead, memory modules can be reserved by one processor in a period of time for one streaming transaction. After the transaction is done, the processor releases the memory module and other processors can acquire the memory module for other transactions. Based on this transaction-level sharing model, a shared memory design does not need to use any buffer. Figure 6.21 shows an example of such bufferless shared memory module with four input ports. This architecture corresponds to the clock source type shown in Figure 6.8(c). The clock is switched by simple multiplexers using the request-grant-release-ack protocol as shown in Figure 6.15. Four processors can request the ownership of the memory module at the same time. Once one processor's request is granted, the memory's clock can be switched to the clock of that processor. The bufferless memory architecture eliminates FIFOs inside the memory modules and results in low latency, high area and energy efficiency.

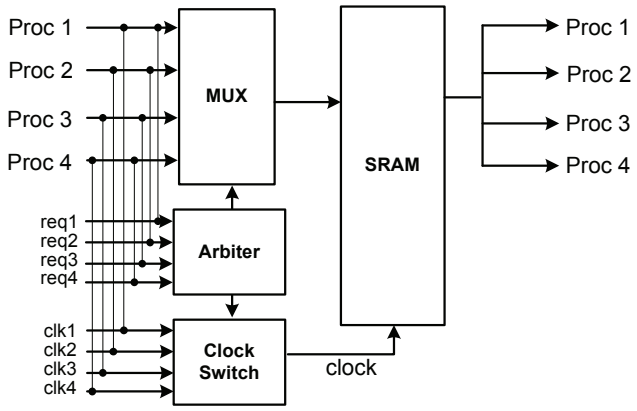



Figure 6.21: A four-port bufferless shared memory module.

### 6.7.2 Micro-architecture

As an example, Figure 6.22 shows the detailed architecture of a bufferless shared memory module with two input ports. The design can be easily extended to support more input ports. A request-grant-release-ack protocol is used to serve multiple requests from processors.

#### Synchronization and Arbitration

The bufferless memory module integrates a small ring oscillator to provide clock to the synchronous hardware mutex. Two-stage or three-stage D flip-flops can be used to synchronize the control signals across the processor and memory clock domains.

A hardware mutex unit is implemented to support mutual exclusive ownership of the memory modules as shown in Figure 6.23. The mutex design is the same as the one used in [131]. The mutual exclusion primitives implement simple test and set locks. The lock is either available, or held by a particular processor during any given cycle. If the priority bits are properly set, the highest priority request is granted if multiple requests arrive at the same time. Otherwise, a least-recently-used policy is adopted to determine which request to serve first. Since the hardware mutex guarantees that only one processor can access the memory at one time, it supports inter-processor synchronization primitive without additional mutex

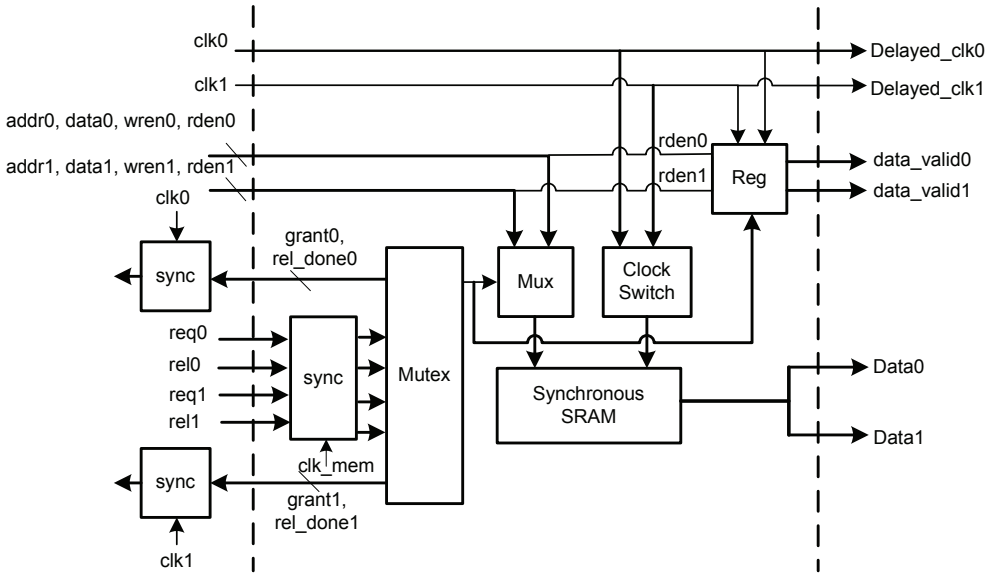



Figure 6.22: Micro-architecture of a two-port source synchronous bufferless shared memory module. For simplicity, the address generator, local ring oscillator and configuration modules are not shown.

like the shared buffered memory module design [131].

### 6.7.3 Performance Evaluation

#### Peak Performance

The memory performance is depicted by two important metrics: throughput and latency. Both bufferless and buffered memory modules achieve a peak throughput of one memory access per cycle. The cycle is one memory cycle for buffered memory and one processor cycle for bufferless memory. The actual throughput depends on maximum clock frequency of the memory module. The bufferless memory module synthesis results (ST 65 nm CMOS and 16 KB single-port SRAM macro) report a maximum clock frequency of 1.37 GHz. At this clock speed, the memory's peak throughput is 21.9 Gbps with 16-bit words.

The worst-case memory latency is for the memory read request. The bufferless memory read latency includes the one-time memory request-grant latency, memory module access

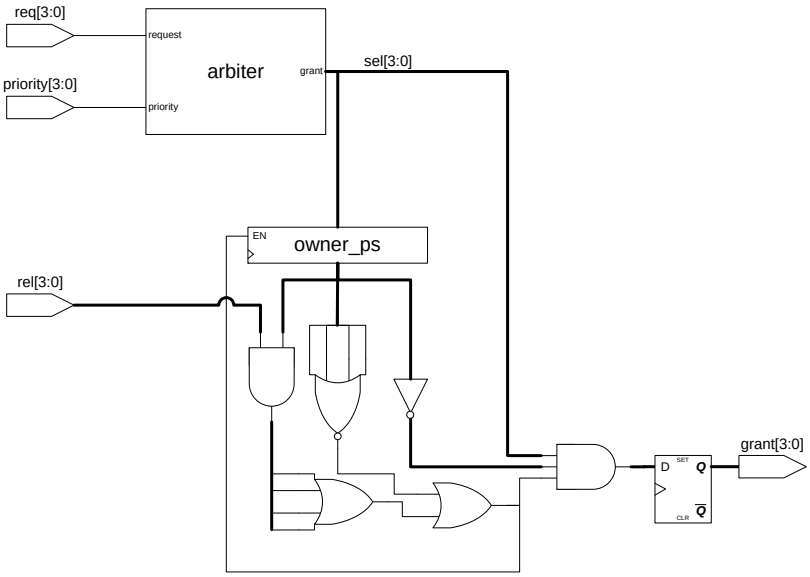



Figure 6.23: Mutual exclusion primitive (mutex) [131]. The mutex implements an atomic test and set lock. The current owner of the mutex is stored in $owner\_ps$. The arbiter manages simultaneous mutex requests.

latency, round-trip interconnection latency and the processor side FIFO write and read synchronization latency. The buffered memory read latency includes the processor memory port latency, memory module input and output FIFO read and write synchronization latency, memory module latency, round-trip interconnection latency and processor pipeline latency for memory access from non-adjacent processors.

The interconnection latency is a variable depending on the distance between processors and memory modules. Let us assume a processor is adjacent to a memory module and there is no request from other processors. The buffered memory latency can be expressed as:

$$L_{proc} = L_{FIFO\text{-}wr} + L_{FIFO\text{-}rd} + L_{mem\text{-}port} \quad (6.6)$$

$$L_{mem} = L_{FIFO\text{-}rd} + L_{mem\text{-}module} + L_{FIFO\text{-}wr} \quad (6.7)$$

$$L_{total} = L_{mem} + L_{proc} \quad (6.8)$$

The FIFO read and write latency depend on the number of pipe stages used to synchro-

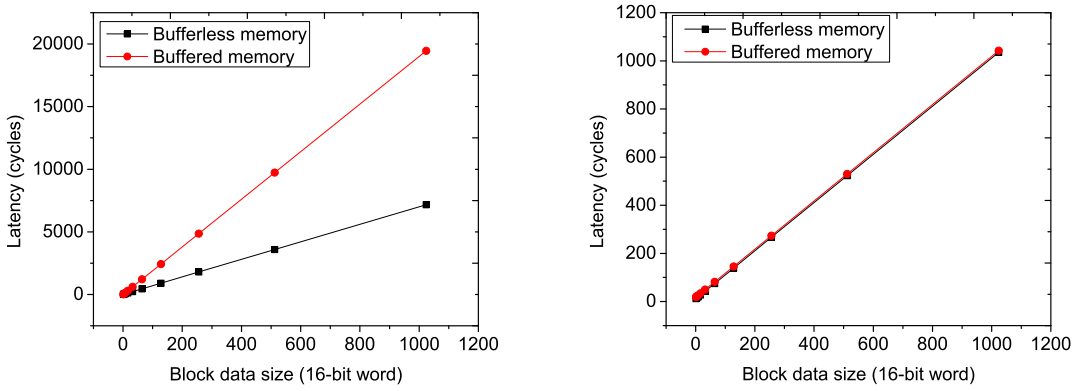



Figure 6.24: Estimated latencies of reading a block of data for buffered and bufferless memory modules (a) non-burst read mode, (b) burst read mode.

nize data across the clock boundary. In this work, two-stage synchronizer is used. Thus, a FIFO takes three cycles for both read and write side. To sum it up, single read latency of a buffered memory is 8 processor cycles and 11 memory cycles. If memories and processors are clocked at the same frequency, this is a minimum latency of 19 cycles. For the burst read mode, after the initial latency of the first word, the rest of words take one cycle per read if all the FIFOs are not full. This cycle can be expressed as: $max(T_{proc}, T_{mem})$.

The bufferless memory latency can be expressed as follows:

$$L_{req} = L_{req\text{-}sync} + L_{mutex} + L_{gnt\text{-}sync} \quad (6.9)$$

$$L_{rel} = L_{rel\text{-}sync} + L_{mutex} + L_{ack\text{-}sync} \quad (6.10)$$

$$L_{read} = L_{mem} + L_{FIFO\text{-}wr} + L_{FIFO\text{-}rd} \quad (6.11)$$

The signal synchronization uses two cycles and the mutex takes one memory cycle. The SRAM read uses two processor cycles. The *request* and *release* take three memory cycles and two processor cycles. After setting up the clock source, the read latency is 8 processor cycles per read. For the burst read mode, after the initial setup latency and the first word read latency, the rest of words takes one processor cycle per word.

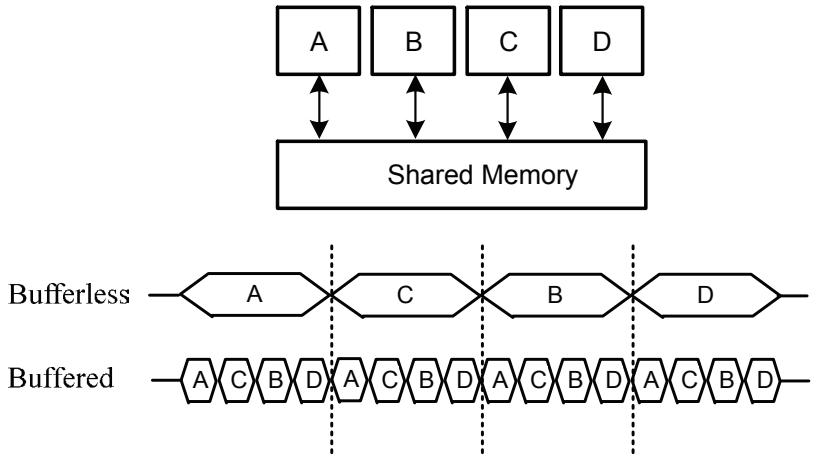



Figure 6.25: Memory bus transactions for four-port buffered and bufferless memory modules.

Based on previous buffered and bufferless memory single read latency calculations, Figure 6.24 shows estimated block data read latencies for buffered and bufferless memory modules. In the non-burst read mode, a bufferless memory module shows a 58% latency reduction compared to buffered memory modules. In the burst read mode, the bufferless memory slightly reduces the access time compared with buffered memory modules, and the time difference is negligible when the block data size is large.

When the memory module is shared by multiple processors, the time to read a block of data in burst mode for bufferless and buffered memory module is close, which depends on the available memory bandwidth. This is illustrated by Figure 6.25 where memory bandwidth is shared in a fine-grained way for buffered memory modules and in a coarse-grained way for bufferless memory modules. Since bufferless memories use clock sources from multiple processors, one processor's memory access time may depend on the other processor's transaction time if they need to wait for the memory bus. Thus, slower processors might drag down the speed of faster processors. In reality, processors running at similar clock frequencies can be tiled together to the same shared memory module. Processors can also be boosted up to the highest clock frequency for memory access so that they do not affect each other.

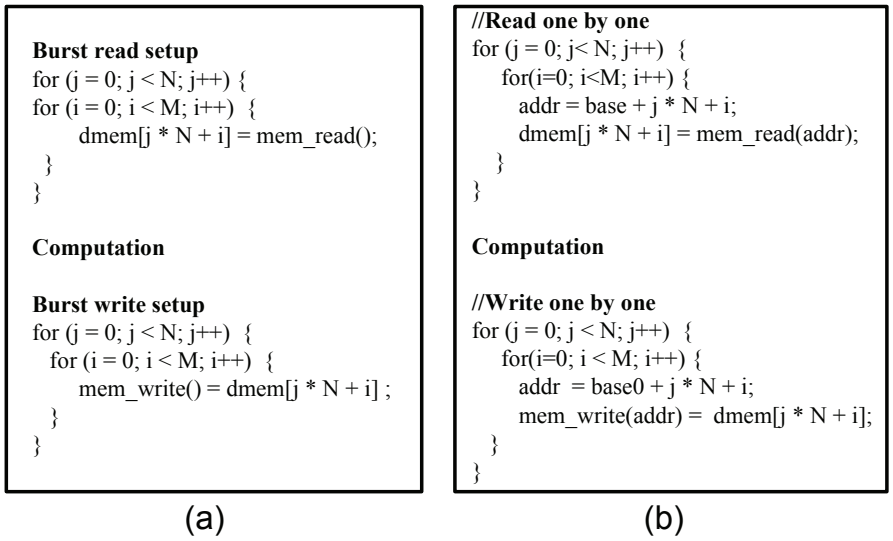



Figure 6.26: Example codes of video application running at one AsAP processor with (a) burst memory access, (b) non-burst memory access.

Table 6.3: Memory requirement and computation workload for typical video encoding tasks in H.264/AVC and HEVC

| **Applications** | Data Input (words) | Data Output (words) | Workload (cycles) |
|---|---|---|---|
| 4x4 Integer Transform | 16 | 16 | 80 |
| 8x8 Integer Transform | 64 | 64 | 320 |
| 4x4 Quantization | 16 | 16 | 336 |
| 4x4 Intra Prediction | 24 | 16 | 708 |
| 16x16 Intra Prediction | 256 | 288 | 11328 |
| 8x8 Sample Adaptive Offset | 84 | 64 | 1664 |

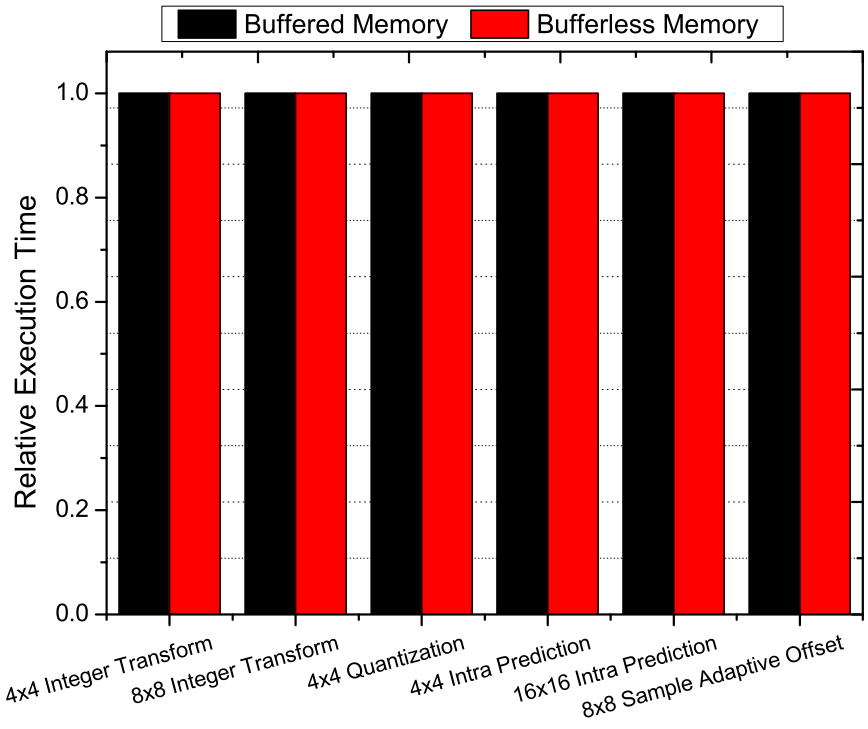
### Application Performance

In order to exploit task-level parallelism, most of video encoding tasks can be partitioned into three phases: reading data from either input processors or shared memories, computation and writing data to output processors or shared memories. Let's assume the tasks use the shared memory as the main data source and destination. Applications can be generated as shown in Figure 6.26 where video data are normally stored in memory as 2D M by N blocks. This model allows us to estimate the effect of memory access time towards the total application performance.

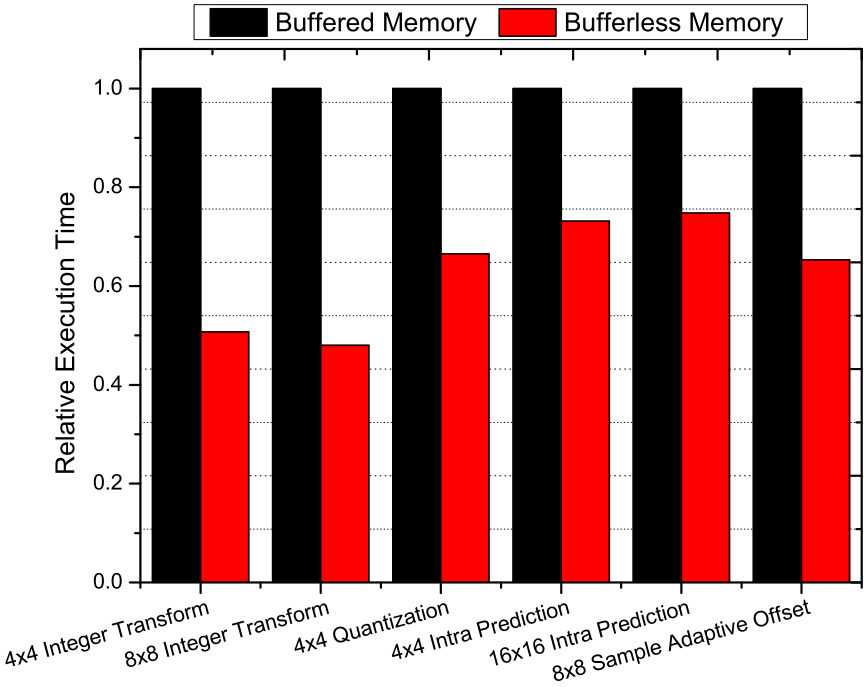
The application performance can be estimated by the number of input data, the computational workload in terms of processor cycles and the number of output data. Table 6.3 lists the characteristics of several key tasks in H.264 and HEVC. The computational workload of H.264 tasks is from actual AsAP2 simulations. The computational workload of Sample Adaptive Offset is estimated from the pseudo assembly coding of SAO on AsAP2. The intra-prediction supports three modes: vertical, horizontal and mean DC.

Let's first consider the case where all of the kernels are mapped onto one processor with one shared memory module and we assume processors and memories run at the same frequency. Figure 6.27 shows the relative application execution time using a bufferless and buffered memory with either burst access mode or non-burst access mode. For burst mode, all applications have similar execution time for buffered and bufferless memory as shown in Figure 6.27(a). For non-burst mode, the bufferless memory reduces 35% to 52% of the total application execution time compared with a buffered memory. The 8 x 8 intra prediction is the most computation intensive tasks, which yields the smallest benefits of replacing a buffered memory by a bufferless memory. There might be chances that the buffered memory can hide some memory read latencies by issuing multiple read addresses and reading data back at different time or inserting instructions between sending an address and reading the data back.

Let's consider a sharing case where multiple copies of the same tasks listed in Table 6.3 are distributed to multiple processors which share one memory module. For burst mode, as we discuss in previous subsection, the relative execution time between bufferless and buffered memory stays the same as non-sharing case. For non-burst mode, the buffered memory has an advantage over bufferless memory that the access latency from multiple processors can overlap between each other. For example, if two processors share one buffered memory, a single read takes 19 cycles for each processor. However, the overall execution time is 20 cycles for the two reads. As for bufferless memory, a single read takes 8 cycles for each processor. However, the overall execution time is 16 cycles for the

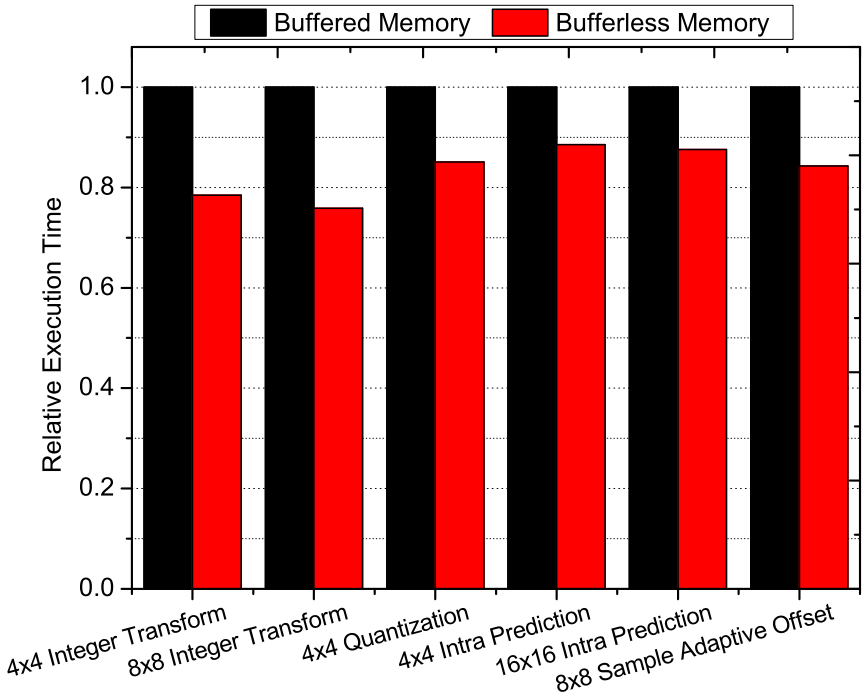



(a)

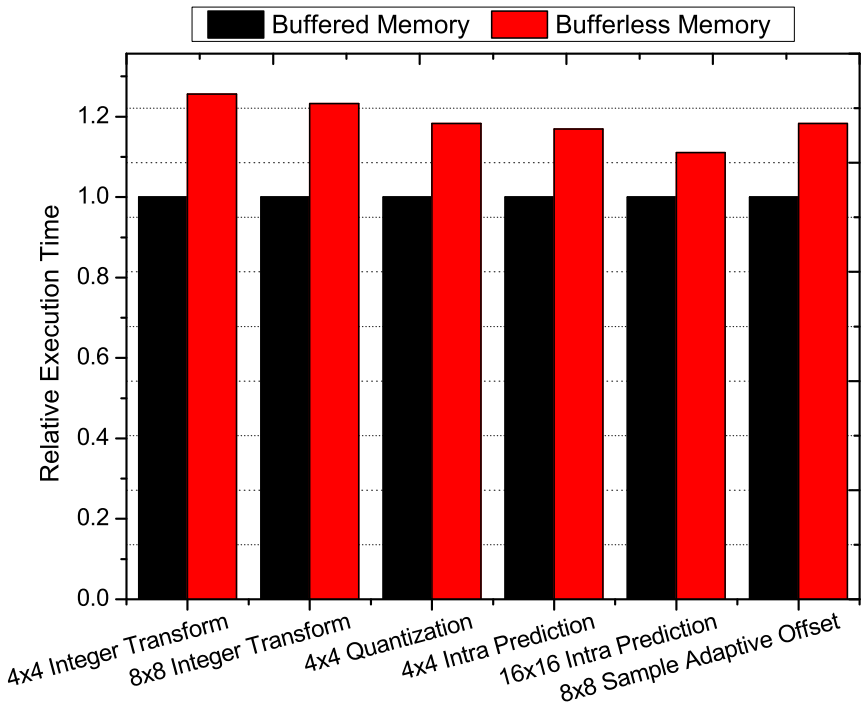



(b)

Figure 6.27: The relative application execution time of a bufferless memory versus a buffered memory without sharing among processors (a) burst mode, (b) non-burst mode.

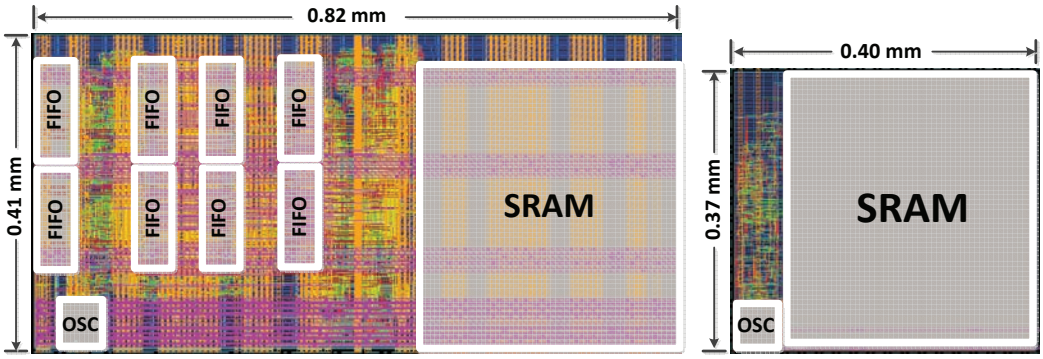



(a)




(b)

Figure 6.28: The relative application execution time of a buffered memory versus a bufferless memory in no-burst mode and (a) two processors, (b) four processors share one memory module.




(a) (b)

Figure 6.29: The DRC and LVS clean layouts of memory modules at 65 nm CMOS technology (a) a 16 KB FIFO buffered shared memory module, (b) a 16 KB bufferless shared memory module.

two reads. Figure 6.28 shows the relative execution time between buffered and bufferless memory in two cases where two processors and four processors share one memory module. When shared by two processors, the bufferless memory module still reduces 11% to 24% overall execution time compared with the buffered memory module as shown in Figure 6.28. When shared by four processors, the execution time increases from 11% to 26% for the bufferless memory compared with the buffered memory. However, this disadvantage of bufferless memory has little effect in video applications where four processors share one memory module with a non-burst mode is rare.

### 6.7.4 Implementation Results

In order to compare with previous 4-port 16 KB FIFO buffered shared memory module, we have implemented a four-port bufferless memory module with a 16 KB single-port SRAM. The primary architecture is shown in Figure 6.21. The bufferless memory module also includes four input ports, which utilize a small state machine to support burst and non-burst memory access. Since the link interface to AsAP processor utilize the processor-processor interconnection network, the link can only provide one 16-bit data per cycle. Thus, it takes two cycles to latch address and data for write operations. Four address gener-

Table 6.4: Layout results of the 4-port 16 KB buffered and bufferless shared memory modules based on 65 nm CMOS at 1.3 V supply voltage and $25°C$

| Memory | Area ($mm^2$) | SRAM Ratio | Max. Freq. (MHz) | Power (mW) |
|---|---|---|---|---|
| Buffered | 0.34 | 37% | 1300 | 50.3 |
| Bufferless | 0.15 | 83% | 1370 | 28.7 |

ators are provided to each input port to support burst access modes. Both the bufferless and buffered memory modules are implemented with a fully automated design flow spanning from RTL description to layout-level verification with STMicroelectronics 65 nm CMOS technology. The memory modules are synthesized from Verilog with Synopsys Design Compiler and laid out with an automatic timing-driven physical design flow with Cadence SoC Encounter. A configurable oscillator (OSC) is manually designed from standard cells and laid out separately.

Figure 6.29(a) shows the final DRC and LVS clean layout of a 16 KB buffered shared memory module composed of eight 32-word dual-clock FIFOs, one configurable oscillator, a 16 KB single-port SRAM macro and other logic circuits. Figure 6.29(b) shows the proposed 16 KB bufferless shared memory module with one configurable oscillator, a 16 KB single-port SRAM macro and other logic circuits.

Table 6.4 shows the layout results of the 16 KB buffered and bufferless memory modules based on 65 nm CMOS technology at 1.3 V and $25°C$. The area of bufferless memory is less than half of the buffered memory area, with a 83% SRAM area utilization ratio. The size of the bufferless memory is slightly smaller than a AsAP2 processor core (0.17 $mm^2$) which makes it easy to be physically integrated into the processor array. The bufferless memory achieves a slightly higher clock frequency and increases the burst-mode throughput by 1% compared with the buffered memory. The power consumption is estimated by assuming both memory and processor clock run at 1070 MHz (maximum frequency of processor core). The activity factor is assumed to be 20% for all inputs. The bufferless memory is much more power efficient than the buffered memory design by reducing 43%

of total power consumption.

## 6.8 Related work

Besides the traditional cache hierarchies for general purpose processors, scratch-pad memories are widely used in embedded systems [132]. The buffered and bufferless memory modules can also be considered as scratch-pad memories with specific features such as being shared by multiple processors in a GALS environment, distributed across the chip and interconnected through a circuit-switch network.

Many commercial multi-core processors have been released. Most of them use traditional cache hierarchies. Intel's latest sandy-bridge processor contains up to 4 processor cores and graphic processing unit (GPU) [107]. Each core has a dedicated two-level cache hierarchy and a 8 MB L3 cache is shared between the cores, as in a traditional shared memory multiprocessor system. Sandy Bridge's ring interconnect fabric connects all the elements of the chip, including the CPUs, the GPU and the L3 cache. The CELL processor is a multiprocessor targeted at multimedia applications [133]. The processor contains a single power processing element (PPE), and eight additional synergistic processing elements (SPE). SPEs adopts a scratch-pad memory architecture which allows each SPE to execute without concern for memory coherency among processors.

Many-core architectures attempt to address the scalability concerns of ever shrinking feature sizes and increasing clock speeds. Tile based architectures, such as MIT's RAW processor [134] and its commercial successor Tilera [135], consist of many uniform processing elements. Each tile is a fully functional CPU and contains a local instruction and data cache. In contrast to traditional systems, this cache may be software managed, or treated as a stand alone memory. Intel presented a 48-core processor called Single-Chip Cloud Computer (SCC) [100]. A total of 24 tiles are connected with a 4x6 2D mesh network. Each tile has two processor cores, each with 16 KB instruction and 16 KB data cache

plus a unified 256 KB L2 cache. The tile also has a 16 KB message passing buffer shared by the two cores.

Transactional memories provide a programming interface which simplifies the parallel programming by guaranteeing that transactions appear to execute atomically, consistently and in isolation [136]. There are hardware and software approaches to implement transactional memory. IBM will ship a first commercial microprocessor supporting hardware transactional memory [137]. Our bufferless memory design acts like the transactional memory. Once an exclusive ownership of the memory is acquired, memory access becomes atomic and in isolation.

The recent development of 3D integration technology enables stacking memory modules directly on top of processors, therefore reducing memory latency and increasing memory bandwidth [138]. In 2007, Intel introduces an experimental 80-core design with stacked memory [139]. This trend is followed by two experimental chips: 3D-MAPS [140] and Centip3De [141] published in 2012. The 3D-MAPS is a 2-layer 3D system that contains 64 customized processor cores with 256 KBs scratch-pad stacked SRAM. The Centip3De is a near-threshold 7-layer 3D system that contains 128 ARM Cortex-M3 cores and 256 MB of stacked DRAM. One of the challenging problem for 3D memory stacking is how to dissipate the heat building up within the stack. The research of 3D memory stacking mainly focuses on tool development, physical design and fabrication.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

This dissertation thoroughly analyzes the H.264 video encoding algorithms. The amount of computation and memory requirement of underlying computation-intensive units have been identified and analyzed. This research suggests that video encoding which is composed of a transformation-based small block data-flow processing is suitable for fine-grained message-passing style many-core architecture.

Then, the dissertation proposes a fine-grained parallel programming methodology and successfully demonstrate fine-grained many-core architecture can achieve high performance and energy efficiency for both video encoding algorithms with high data-level parallelism like integer transform and quantization and serial algorithms with fine-grained task-level parallelism like CAVLC. The proposed programming methodology yields an H.264/AVC residual encoder capable of realtime 1080p (1920x1080) HDTV encoding with both higher energy efficiency and area efficiency compared with other software approaches in common DSPs and customized hybrid multi-core architectures.

Next, this dissertation proposes seven low area overhead and low design complexity topologies other than the commonly-used 2D mesh for dense on-chip networks. The

proposed topologies include two 8-neighbor meshes, two 5-nearest-neighbor and three 6-nearest-neighbor topologies—three of which use a novel house-shaped and hexagonal-shaped tile. Two complete applications are mapped onto all topologies for realistic comparisons. Commonly available commercial CAD tools are used to implement tiled CMPs for all proposed topologies including the two non-rectangular processor tiles. The application mapping and chip implementation results demonstrate the effectiveness of the inter-processor interconnect of all proposed topologies. Compared with 2D mesh, the hexagonal-shaped 6-nearest-neighbor topology reduces 22% application area and 17% average power consumption with a 2.9% area increase per processor tile. The rectangular-shaped 6-nearest-neighbor topology provides the same interconnect architecture as the hexagonal-shaped tile. Despite being less power-efficient, its simpler physical design makes it an attractive design alternative for many-core dense on-chip networks.

Motivated by the fact that video encoding tasks normally read and write a block of data at one time in one transaction, the third part of this dissertation proposes a novel source synchronous bufferless shared memory to enable safe memory sharing among multiple processors with different clock domains. Compared with the previous FIFO buffered memory design, the bufferless memory achieves lower latency, higher throughput, lower area overhead and lower power consumption. The bufferless memory also supports direct communication with far-away processors through the existing processor-processor circuit switch interconnection network. The implementation results shows that a 16 KB bufferless memory module reduces 58% single memory access latency and has slightly higher throughput (1%) in a burst mode compared to the 16 KB buffered memory module. The bufferless memory module also reduces the area overhead from 63% to 17% compared with buffered memory module, which yields a power reduction by 43%.

## 7.2 Future Work

There are quite a few interesting research topics on many-core processors for video and DSP applications which are worthwhile for further investigation.

- **High Efficiency Video Coding (HEVC)** The High Efficiency Video Coding (HEVC) also called H.265, is the successor of H.264/AVC. The new standard's committee draft is approved in February 2012. The new standards include some new features such as variable block size (coding unit), new loop filters such as SAO (Sample Adaptive Offset) and wider pixel bit width 10 to 12 bits. HEVC achieves over 40% bitrate reduction compared with H.264/AVC. However, the coding efficiency comes at the cost of higher computational complexities, which brings further challenges for parallel programming. More research is required to explore the new possibility of mapping HEVC to the fine-grained many-core computation platform.

- **Automatic Mapping Tool** As the application complexity increases, the number of small tasks increases accordingly. The number of tasks has reached over 100 for H.264 baseline encoder and this number may grow to hundreds or even thousands for future video applications. The manual application mapping is not feasible in this case. The automatic mapping tool should provide a capability to reduce the number of processors, the total interconnection link length and power consumption. The tool should also be aware of the heterogenous components such as shared memories and accelerators. The automatic mapping tool should also support the non-2D-mesh topologies such as the 6-neighbor hex topology proposed in this work.

- **Reconfigurable Accelerators** Some video encoding tasks such as CABAC and de-block filtering are not efficient when mapped to AsAP system. The coding blocks may vary by standards. Building accelerators for these blocks is time-consuming. There might be new possibility to build some reconfigurable fabrics for these tasks.

- **Memory Interconnections** The proposed shared memory system uses source synchronous static circuit-switch network for long-distance communication. A dedicate link needs to be assigned for a particular transaction. This is not efficient in the case where memory traffic is too sparse to saturate the link bandwidth. A low overhead router with cut-through packet forwarding capability may be helpful in this case. A hybrid architecture with both circuit-switch network and dynamic routing may be the future direction for memory interconnections for video encoding tasks.

# Glossary

**AsAP** For *Asynchronous Array of simple Processors.* A parallel DSP processor consisting of a 2-dimensional mesh array of very simple CPUs clocked independently with each other.

**Arbiter** A circuit module which handles multiple access requests to a shared resource and grants the access permission for one of these requests preventing them to simultaneously access the shared resource.

**AsAP2** The second generation of AsAP chips which also includes a few specific accelerators (FFT, Viterbi, Motion Estimation) and shared memory modules. It has a reconfigurable source synchronous network supporting long-distance interconnects for processors. Per-core DVFS is also supported for dynamic power savings.

**CABAC** For *Context Adaptive Binary Arithmetic Coding*, an entropy encoding method in H.264/AVC main and high profile.

**CAVLC** For *Context Adaptive Variable Length Coding*, an entropy encoding method in H.264/AVC baseline profile.

**CMP** For *Chip Multi-processor*, a computer architecture which integrates multiple processors into a single chip to improve processor performance.

**CPI** For *Cycles-per-instruction.* Normally the CPI for pipelined processor is larger than 1 due to the pipeline hazard or missed Cache fetch.

**DCT** For *Discrete Cosine Transform*, it is used to transforms a signal or image from the spatial domain to the frequency domain.

**DRAM** For *Dynamic Random Access Memory*. A type of memory that it need to be refreshed periodically. It is slower but more compact than the static RAM.

**DSP** For *digital signal processing or the processors for DSP.*

**FFT** For *Fast Fourier Transform*, an efficient algorithm to compute the discrete Fourier transform and its inverse.

**FIFO** For *FIFO First-In First-Out*. A buffer queue with in-order operations: the word which is written in to the buffer first will be read out of the queue first.

**FO4** For *Fanout 4.* A method to define the circuit delay using the delay of an inverter with 4 inverters load.

**GALS** For *Globally Asynchronous Locally Synchronous*. A design methodology in which major design blocks are synchronous, but interface to other blocks asynchronously.

**GDSII** For *Graphic Database System II*. A database file format which is the de facto industry standard for data exchange of integrated circuit or IC layout artwork.

**H.264/AVC** A standard for video compression. It is also known as MPEG-4 part 10.

**HEVC** An under-drafted video compression standard known as high efficiency video coding.

**Mbps** For *Megabit per second*, a unit of data transfer rate.

**MTBF** For *Mean Time Between Failures*, a common measures of reliability.

**ME** For *Motion Estimation*, is the process of determining motion vectors that describe the difference between one 2D image and another.

**NoC** For *Network on Chip*. An on-chip communication architecture which communicates between modules in the chip using switches/routes, as in the network.

**RTL** *Register-Transfer Level*. RTL language is a hardware description language used to model and simulate hardware modules at the gate and register level. A hardware module modeled in the RTL level could be synthesized to a netlist of CMOS cell gates used for chip layout. Two most-used RTL languages are Verilog and VHDL.

**Scratchpad Memory** An on-chip memory with independent address space for temporary data storage.

**SAD** For *Sum of Absolute Differences*. A widely used simple algorithm for measuring the similarity between image blocks.

**SIMD** For *Single Instruction, Multiple Data*. A data parallelism technique where one single instruction can execute multiple data in parallel.

**SRAM** For *Static Random Access Memory*. A type of memory that is faster and more reliable than the more common DRAM (dynamic RAM). The term static is derived from the fact that it does not need to be refreshed like dynamic RAM.

**Viterbi decoder** An algorithm to decode a bitstream that has been encoded using forward error correction based on a convolutional code, developed by Andrew J. Viterbi in 1967.

**VLIW** For *Very long instruction word*, a computer architecture which fetches multiple independent instructions at the same clock cycle to execute them in parallel, to improve the system performance.

# Related publications


1. **Zhibin Xiao**, Bevan Baas, Processor Shapes and Topologies for Compact Processor Tiles and Dense On-Chip Networks, journal paper under review.

2. **Zhibin Xiao** and Bevan Baas, A Hexagonal Shaped Processor and Interconnect Topology for Tightly-tiled Many-core Architecture, to appear in the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC) , Santa Cruz, CA, Oct. 2012.

3. **Zhibin Xiao**, Bevan Baas, Processor Shapes and Topologies for Compact Processor Tiles and Dense On-Chip Networks, *IEEE International Solid-Sate Circuit Conference (ISSCC 2012) Student Forum*, San Francisco, CA, Feb. 2012.

4. **Zhibin Xiao**, Stephen Le, Bevan Baas, A Fine-Grained Parallel Implementation of a H.264/AVC Encoder on a 167-processor Computational Platform, *IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Pacific Grove, CA, November 2011.

5. **Zhibin Xiao**, Bevan Baas, A 1080p H.264/AVC Baseline Residual Encoder for a Fine-grained Many-core System, *IEEE Transactions on Circuit and Systems for Video Technology*, vol. 21, no. 7, pp. 890–902, July 2011.

6. **Zhibin Xiao**, Stephen Le, An Energy-efficient Parallel H.264/AVC Baseline Encoder on a Fine-grained Many-core System, *SRC Technology and Talent for the 21st Century (TECHCON)*, Sep. 2010.

7. Dean N. Truong, Wayne H. Cheng, Tinoosh Mohsenin, Zhiyi Yu, Anthony T. Jacobson, Gouri Landge, Michael J. Meeuwsen, Christine Watnik, Anh T. Tran, **Zhibin Xiao**, Eric W. Work, Jeremy W. Webb, Paul V. Mejia, Bevan M. Baas, A 167-Processor Computational Platform in 65 nm CMOS, *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 44, no. 4, pp. 1130–1144, April 2009.

8. **Zhibin Xiao**, Bevan Baas, A High-Performance Parallel H.264 CAVLC Encoder on a Fine-Grained Many-core System, *International Conference on Computer Design (ICCD)*, Sep. 2008.

9. Dean Truong, Wayne Cheng, Tinoosh Mohsenin, Zhiyi Yu, Toney Jacobson, Gouri Landge, Michael Meeuwsen, Christine Watnik, Paul Mejia, Anh Tran, Jeremy Webb, Eric Work, **Zhibin Xiao**, Bevan Baas, A 167-processor Computational Array for Highly-Efficient DSP and Embedded Application Processing, *IEEE HotChips Symposium on High-Performance Chips(HotChips 2008)*, August 2008.

10. Dean Truong, Wayne Cheng, Tinoosh Mohsenin, Zhiyi Yu, Toney Jacobson, Gouri Landge, Michael Meeuwsen, Christine Watnik, Paul Mejia, Anh Tran, Jeremy Webb, Eric Work, **Zhibin Xiao**, Bevan Baas, A 167-processor 65 nm Computational Platform with Per-Processor Dynamic Supply Voltage and Dynamic Clock Frequency Scaling, in proceedings of the *Symposium on VLSI Circuits*, June 2008.

# Bibliography


[1] B. G. Haskell, P. G. Howard, et al. Image and video codingłemerging standards and beyond. *IEEE Trans. Circuits Syst. Video Technol.*, 8(7):814–837, Nov 2006.

[2] S. Borkar. Low power design challenges for the decade. *Asia and South Pacific Design Automatic Conference (ASP-DAC)*, pages 293–296, 2001.

[3] Rusu Stefan, Tam Simon, Muljono Harry, Stinson Jason, Ayers David, Chang Jonathan, Varada Raj, Ratta Matt, Kottapalli Sailesh, and Vora Sujal. A 45 nm 8-core enterprise xeon processor. *Journal of Solid-State Circuits*, 45(1):7–14, Jan. 2010.

[4] Kurd Nasser A., Bhamidipati Subramani, Mozak Christopher, Miller Jeffrey L., Wilson Timothy M., Nemani Mahadev, and Chowdhury Muntaquim. Westmere: A family of 32nm ia processors. In *Proc. of IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pages 96–97, Feb. 2010.

[5] Satish Damaraju, George Varghese, Sanjeev Jahagirdar, Tanveer Khondker, Robert Milstrey, Sanjib Sarkar, Scott Siers, Israel Stolero, and Arun Subbiah. A 22nm ia multi-cpu and gpu system-on-chip. In *Proc. of IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pages 56–57, Feb. 2012.

[6] J. Stinson and S. Rusu. A 1.5 GHz third generation Itanium processor. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 252–253, Feburary 2003.

[7] S. Naffziger, T. Grutkowski, and B. Stackhouse. The implementation of a 2-core multi-threaded Itanium family processor. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 182–183, Feburary 2005.

[8] S. Rusu, S. Tam, H. Muljono, D. Ayers, , and J. Chang. A 65nm dual-core multi-threaded Xeon processor with 16MB L3 cache. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 102–103, Feburary 2006.

[9] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles. A 65nm 2-billion-transistor quad-core Itanium processor. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 92–598, Feburary 2008.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture, A Quantitative Approach*, chapter Memory Hierarchy Design. Morgan Kaufmann, San Francisco, CA, third edition, 2003.

[11] Zhiyi Yu, Michael Meeuwsen, Ryan Apperson, Omar Sattari, Michael Lai, Jeremy Webb, Eric Work, Tinoosh Mohsenin, Mandeep Singh, and Bevan M. Baas. An asynchronous array of simple processors for DSP applications. In *IEEE International Solid-State Circuits Conference, (ISSCC '06)*, pages 428–429, Feb. 2006.

[12] R. Bhargava, R. Radhakrishnan, B. Evans, and L. John. Characterization of MMX-enhanced DSP and multimedia applications on a general purpose processor. In *Digest of the Workshop on Performance Analysis and Its Impact on Design held in conjunction with ISCA98*, pages 16–23, 1998.

[13] R. Bhargava et al. Evaluating MMX technology using DSP and multimedia applications. *IEEE Micro*, pages 37–46, Dec. 1998.

[14] J. Fritts, W. Wolf, and B. Liu. Understanding multimedia application characteristics for designing programmable media processors. In *SPIE Photonics West, Media Processors'99,*, pages 2–13, San Jose, CA, Jan. 1999.

[15] J. Fritts et al. Performance of image and video processing with general-purpose and media ISA extensions. In *International Symposium on Computer Architecture*, pages 124–135, May 1999.

[16] H. Nguyen and L. K. John. Exploiting SIMD parallelism in DSP and multimedia algorithm using the AltiVec technology. In *International Conference on Supercomputing*, pages 11–20, May 1999.

[17] Zhiyi Yu. *High Performance and Energy Efficient Multi-core Systems for DSP Applications*. PhD thesis, University of California Davis, Davis,CA, Sep. 2007.

[18] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. Mejia, and B. M. Baas. A 167-processor computational platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4):1130–1144, April 2009.

[19] Y. W. Huang et al. A 1.3TOPS H.264/AVC single-chip encoder for hdtv applications. In *IEEE International Solid-State Circuits Conference, (ISSCC '06)*, pages 128–130, Feb. 2006.

[20] C. C. Lin et al. A 160kgate 4.5kb SRAM H.264 video decoder for HDTV applications. In *IEEE International Solid-State Circuits Conference, (ISSCC '06)*, pages 406–407, Feb. 2006.

[21] Hsiu-Cheng Chang et al. A 7mw-to-183mw dynamic quality-scalable H.264 video encoder chip. In *IEEE International Solid-State Circuits Conference, (ISSCC '07)*, pages 280–281, Feb. 2007.

[22] Yu-Kun Lin et al. A 242mw 10mm$^2$ 1080p H.264/AVC high-profile encoder chip. In *IEEE International Solid-State Circuits Conference, (ISSCC '08)*, pages 314–615, Feb. 2008.

[23] Zhenyu Liu et al. A 1.41w h.264/avc real-time encoder soc for hdtv1080p. In *Symposium on VLSI Circuits, (VLSI '07)*, June 2007.

[24] Tung-Chien Chen et al. 2.8 to 67.2mw low-power and power-aware h.264 encoder for mobile applications. In *Symposium on VLSI Circuits, (VLSI '07)*, June 2007.

[25] Koyo Nitta et al. An h.264/avc high422 profile and mpeg-2 422 profile encoder lsi for hdtv broadcasting infrastructures. In *Symposium on VLSI Circuits, (VLSI '08)*, June 2008.

[26] Kenichi Iwata et al. A 256mw full-hd h.264 high-profile codec featuring dual macroblock-pipeline architecture in 65nm cmos. In *Symposium on VLSI Circuits, (VLSI '08)*, June 2008.

[27] DSP Products,C6x Information, Texas Instruments. *Fixed- and Floating-Point DSP-SłOne Architecture)*, 1998.

[28] P. Kalapathy. Hardware-software interactions on mpact. *IEEE Micro*, 17:20–26, 1997.

[29] S. Rathnam and G. Slavenburg. An architectural overview of the programmable multimedia processor, tm-1. In *Proc. Compcon*, pages 319–326, 1996.

[30] Ruby Lee. Accelerating multimedia with enhanced microprocessors. *IEEE Micro*, 15:22–32, 1995.

[31] Alex Peleg and Uri Weiser. Mmx technology extension to the intel architecture. *IEEE Micro*, 16(4):42–50, 1996.

[32] S. K. Raman, V. Pentkovski, and J. Keshava. Implementing streaming simd extensions on the pentium iii processor. *IEEE Micro*, 20(4):28–39, 2000.

[33] R.B. Lee. Subword parallelism with max-2. *IEEE Micro*, 16(4):51–59, 1996.

[34] M. Tremblay, J.M. OConnor, V. Narayanan, and L. He. Vis speeds new media processing. *IEEE Micro*, 16(4):10–20, 1996.

[35] M. Phillip et al. Altivec technology: Accelerating media processing across the spectrum. In *Proc. HOTCHIPS10*, Aug. 1998.

[36] D. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling. Architecture design of reconfigurable pipelined datapaths. In *Proc. 20th Anniversary Conf. Advanced Research in VLSI*, pages 23–40, Feb. 1997.

[37] H. Singh et al. Morphosys: An integrated reconfigurable architecture. In *Proc. NATO Symp. Systems Concepts and Integration*, Feb. 1998.

[38] V. Baumgarte et al. Pact xppła self-reconfigurable data processing architecture. In *Proc. Eng. of Reconfigurable Systems and Algorithms,(ERSA2001)*, Feb. 1998.

[39] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucek Khailany, Abelardo Lpez-lagunas, Peter R. Mattson, and John D. Owens. A bandwidth-efficient architecture for media processing. In *In 31st International Symposium on Microarchitecture*, pages 3–13, 1998.

[40] Matthew Drake, Henry Hoffman, Rodric Rabbah, and Saman Amarasinghe. Mpeg-2 decoding in a stream programming language. In *In International Symposium on Computer Architecture (ipdps)*, Rhodes Island, Greece, Apr. 2006.

[41] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *In International Symposium on Computer Architecture (ISCA)*, pages 161–171, June 2000.

[42] S. Ciricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, , and A. Saidi. The reconfigurable streaming vector processor (rsvp$^{TM}$). In *Proc. Int. Symp. Microarchitecture*, pages 141–150, Feb. 2003.

[43] Antonio Gentile and D. Scott Wills. Portable video supercomputing. *IEEE Trasaction on Computers*, 53(8):960–973, Aug. 2004.

[44] Brucek Khailany et al. A programmable 512 GOPS stream processor for signal, image, and video processing. In *IEEE International Solid-State Circuits Conference, (ISSCC '07)*, pages 272–273, Feb. 2007.

[45] B. Flachs, S. Asano, S. H. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano. A streaming processing unit for a CELL processor. In *IEEE International Solid-State Circuits Conference, (ISSCC '05)*, pages 134–135, Feb. 2005.

[46] Mike Butts. Synchronization through communication in a massively parallel processor array. *IEEE Micro*, 27(5):32–40, 2007.

[47] S. Bell et al. TILE64$^{TM}$ processor: A 64-core soc with mesh interconnect. In *IEEE International Solid-State Circuits Conference, (ISSCC '08)*, pages 88–89, Feb. 2008.

[48] M. Nakajima et al. A 40 GOPS 250 mw massively parallel processor based on matrix architecture. In *IEEE International Solid-State Circuits Conference, (ISSCC '06)*, pages 410–411, Feb. 2006.

[49] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Trans. Circuits Syst. Video Technol.*, 13(7):560–576, 2003.

[50] A. Joch et al. Performance comparison of video coding standards using lagrangian coder control. In *Proc. IEEE Int. Conf. on Image Processing*, pages 501–504, 2002.

[51] Lai-Man Po and Wing-Chung Ma. A novel four-step search algorithm for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):313–317, jun 1996.

[52] JVT. H.264/AVC reference software version jm 12.4.

[53] Chung-Cheng Lou, Szu-Wei Lee, and C.-C.J. Kuo. Adaptive motion search range prediction for video encoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(12):1903–1908, Dec. 2010.

[54] Jia-Ching Wang, Jhing-Fa Wang, Jar-Ferr Yang, and Jang-Ting Chen. A fast mode decision algorithm and its vlsi design for h.264/avc intra-prediction. *IEEE Transaction on Circuits and Systems for Video Technology*, 17(10):1414–1422, 2007.

[55] Henrique S. Malvar, Antti Hallapuro, Marta Karczewicz, and Louis Kerofsky. Low-complexity transform and quantization in H.264/AVC. *IEEE Transaction on Circuits and Systems for Video Technology*, 13(7):598–603, 2003.

[56] Dongming Zhang et al. Complexity controllable dct for real-time h.264 encoder. *Journal of Visual Communication and Image Representation*, 18(1):59–67, 2007.

[57] Chung-Ming Chen and Chung-Ho Chen. Complexity controllable dct for real-time h.264 encoder. *IEICE Trans. on Inf. and Syst*, E90-D(1):99–107, 2007.

[58] Detlev Marpe, Heiko Schwarz, and Thomas Wiegand. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions On Circuits and Systems for Video Technology*, 13(7):620–636, 2003.

[59] Yen-Kuang Chen et al. Towards efficient multi-level threading of h.264 encoder on intel hyper-threading architectures. In *Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.

[60] Michael Roitzsch. Slice-balancing H.264 video encoding for improved scalability of multicore decoding. In *Proc. of the 7th ACM and IEEE International Conference on Embedded software*, pages 269–278, 2007.

[61] A. Rodrłguez* et al. Hierarchical parallelization of an h.264/avc video encoder. In *Proc. of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, 2006.

[62] Zhuo Zhao and Ping Liang. Performance comparison of video coding standards using lagrangian coder control. In *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages V 489–492, 2006.

[63] Shuwei Sun, Dong Wang, and Shuming Chen. A highly efficient parallel algorithm for h.264 encoder based on macro-block region partition. *Lecture Notes In Computer Science*, pages 577–585, 2007.

[64] T. Hoare. ”communicating sequential processes”. *Comm. ACM*, 8(21):666–677, 1978.

[65] Ujval Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. The Imagine stream processor. In *Proceedings 2002 IEEE International Conference on Computer Design*, pages 282–288, Sep. 2002.

[66] M. Taylor et al. A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 170–171, Feb. 2003.

[67] B.K. Khailany, T. Williams, J. Lin, E.P. Long, M. Rygh, D.W. Tovey, and W.J. Dally. A programmable 512 GOPS stream processor for signal, image, and video processing. *IEEE Journal of Solid-State Circuits*, 43(1):202–213, Jan. 2008.

[68] Nagai-Man Cheung, Xiaopeng Fan, Oscar C. Au, and Man-Cheung Kung. Video coding on multicore graphics processors. *IEEE Signal Processing Magazine*, 27(2):79–89, Mar. 2010.

[69] Wei-Nien Chen and Hsueh-Ming Hang. H.264/AVC motion estimation implmentation on compute unified device architecture (CUDA). In *IEEE International Conference on Multimedia and Expo*, pages 697–70, April 2008.

[70] C. D. Chien et al. A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications. In *IEEE Int. Sym. on Circuits and Systems (ISCAS)*, pages 3838–3841, May 2006.

[71] Choudhury A. Rahman and Wael Badawy. CAVLC encoder design for real-time mobile video applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(10):873–877, Oct. 2007.

[72] Zhiyi Yu, M.J. Meeuwsen, R.W. Apperson, O. Sattari, M. Lai, J.W. Webb, E.W. Work, D. Truong, T. Mohsenin, and B.M. Baas. AsAP: An asynchronous array of simple processors. *IEEE Journal of Solid-State Circuits*, 43(3):695–705, Mar. 2008.

[73] Dean Truong, Wayne Cheng, Tinoosh Mohsenin, Zhiyi Yu, Toney Jacobson, Gouri Landge, Michael Meeuwsen, Christine Watnik, Paul Mejia, Anh Tran, Jeremy Webb, Eric Work, Zhibin Xiao, and Bevan M. Baas. A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling. In *Symposium on VLSI Circuits, (VLSI ’08)*, June 2008.

[74] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. Mejia, and B. M. Baas. A 167-processor computational platform in 65 nm cmos. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4):1130–1144, April 2009.

[75] Francesco Vitullo, Nicola E., Esa Petri, Sergio Saponara, Luca Fanucci, Michele Casula, Riccardo Locatelli, and Marcello Coppola. Low-complexity link microarchitecture for mesochronous communication in Networks-on-Chip. *IEEE TRANSACTIONS ON COMPUTERS*, 57(9):1196–1201, Sep. 2008.

[76] Sanjive Agarwala et al. A 600-MHz VLIW DSP. *IEEE Journal of Solid-State Circuits*, 37(11):1532–1544, Nov 2002.

[77] G.Bjontegaard and K.Lillevold. Context-adaptive VLC(CVLC) coding of coefficients. Doc.JVT C028r1.doc, May 2002.

[78] Zhibin Xiao and Bevan M. Baas. A high-performance parallel CAVLC encoder on a fine-grained many-core system. In *International Conference on Computer Design, (ICCD '08)*, pages 248–254, October 2008.

[79] Eric W. Work. Algorithms and software tools for mapping arbitrarily connected tasks onto an asynchronous array of simple processors. Master's thesis, University of California, Davis, CA, USA, September 2007. `http://www.ece.ucdavis.edu/vcl/pubs/theses/2007-4`.

[80] Zhiyi Yu and Bevan M. Baas. A low-area interconnect architecture for chip multiprocessors. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2857–2860, May 2008.

[81] Wei Zhao and Yu Cao. New generation of predictive technology model for sub-45nm design exploration. In *ISQED '06: Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 585–590, Mar. 2006.

[82] J.M.Rabaey. *Digital Integrated Circuits – A Design Perspective*. Prentice-Hall International, Inc, second edition, 2003.

[83] W.I.Choi et al. Fast motion estimation with modified diamond search for variable motion block sizes. *IEEE Trans. on Image Processing*, 3:371–374, Sep. 2003.

[84] Li Zhuo, Qiang Wang, et al. Optimization and implementation of H.264 encoder on DSP platform. In *IEEE Int. Conf. on Multimedia and Expo (ICME)*, pages 232–235, July 2007.

[85] Shashi Kant et al. Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications. In *Int. Conf. on Consumer Electronics (ICCE)*, pages 93–94, Jan. 2006.

[86] Xun He, Xiangzhong Fang, Ci Wang, and S. Goto. Parallel HD encoding on CELL. In *IEEE International Symposium on Circuits and Systems (ISCAS 09)*, pages 1065–1068, May 2009.

[87] S. Seo, M. Woh, S. Mahlke, T. Mudge, S. Vijay, and C. Chakrabarti. Customizing wide-SIMD architectures for H.264. In *SAMOS'09: Proceedings of the 9th international conference on Systems, architectures, modeling and simulation*, pages 172–179, 2009.

[88] Intel official website. Intel processor specifications, Oct. 2010. http://ark.intel.com/Product.aspx?id=35569.

[89] Hsiu-Cheng Chang, Jia-Wei Chen, Ching-Lung Su, Yao-Chang Yang, Yao Li, Chun-Hao Chang, Ze-Min Chen, Wei-Sen Yang, Chien-Chang Lin, Ching-Wen Chen, Jinn-Shan Wang, and Jiun-In Quo. A 7mw-to-183mw dynamic quality-scalable h.264 video encoder chip. In *IEEE International Solid-State Circuits Conference*, pages 280–603, Feb. 2007.

[90] Mike Butler. AMD Bulldozer Core - a new approach to multithreaded compute performance for maximum efficiency and throughput. In *IEEE HotChips Symposium on High-Performance Chips (HotChips 2010)*, Aug. 2010.

[91] M. Taylor et al. The design and implementation of a first-generation CELL processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 184–185, Feb. 2005.

[92] M. Horowitz R. Ho, K. Mai. The future of wires. *Proc. of IEEE*, 89:490–504, Apr. 2001.

[93] Zhiyi Yu and B.M. Baas. A low-area multi-link interconnect architecture for GALS chip multiprocessors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(5):750–762, may. 2010.

[94] Erno Salminen, Ari Kulala, and Timo D. Hämäläinen. Survey of Network-on-Chip proposals. *Open Core Protocol International Partnership (OCP-IP): White Paper*, p.1, 2008. [online] Available: http://www.ocpip.org/socket/whitepapers.

[95] Hui Zhang, Marlene Wan, V. George, and J. Rabaey. Interconnect architecture exploration for low-energy reconfigurable single-chip dsps. In *Proc. IEEE Computer Society Workshop On VLSI*, pages 2–8, 1999.

[96] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Effect of traffic localization on energy dissipation in NoC-based interconnect. In *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, pages 1774–1777, 2005.

[97] J. Kim, J. Balfour, and W.J. Dally. Flattened butterfly topology for on-chip networks. *Computer Architecture Letters*, 6(2):37–40, Feb. 2007.

[98] J. Balfour and W.J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of the 20th Annual International Conference on Super Computing*, pages 187–198, 2006.

[99] Vangal S. et al. An 80-Tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 100–101, Feb. 2007.

[100] J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173 –183, Jan. 2011.

[101] Hongyu Chen et al. The y architecture for on-chip interconnect: analysis and methodology. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):588–599, April 2005.

[102] Feng Zhou, Esther Y. Cheng, Bo Yao, Chung-Kuan Cheng, and Ronald Graham. A hierarchical three-way interconnect architecture for hexagonal processors. In *SLIP '03: Proceedings of the 2003 international workshop on System-level interconnect prediction*, pages 133–139, 2003.

[103] Kang G. Shin. Harts: A distributed real-time architecture. *IEEE Computer*, 24(5):25–35, 1991.

[104] Catherine Decayeux and Davis Seme. 3d hexagonal network: modeling, topological properties, addressing scheme, and optimal routing algorithm. *IEEE Trans. on Parallel and Distributed Systems*, 16(9):875–884, Sep. 2005.

[105] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli. A continuous-time hexagonal field-programmable analog array in 0.13um CMOS with 186MHz GBW. In *IEEE International Solid-State Circuits Conference, (ISSCC '08)*, pages 70–71, Feb. 2008.

[106] Allen D. Malony. Regular processor arrays. In *the 2nd Symposium on the Frontiers of Massively Parallel Computation*, pages 499–502, 1988.

[107] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts. A fully integrated multi-CPU, GPU and memory controller 32nm processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 264–266, Feb. 2011.

[108] S. Wong et al. Modeling of interconnect capacitance, delay, and crosstalk in VLSI. *IEEE Trans. Semiconduct. Manufact.*, 13:108–111, February 2000.

[109] PTM. Predictable technology model, interconnect section. Online. http://www.eas.asu.edu/ptm/.

[110] ITRS. International technology roadmap for semiconductors, 2010 update, interconnect section. Online. http://www.itrs.net/reports.html.

[111] Yulei Zhang, James F. Buckwalter, and Chung-Kuan Cheng. Performance prediction of throughput-centric pipelined global interconnects with voltage scaling. In *Proceedings of the 12th ACM/IEEE international workshop on System level interconnect prediction*, SLIP '10, pages 69–76, 2010.

[112] Zhibin Xiao and Bevan Baas. A 1080p H.264/AVC baseline residual encoder for a fine-grained many-core system. *IEEE Transaction on Circuits and Systems for Video Technology*, 21(7):890–902, 2011.

[113] Anh T. Tran, Dean N. Truong, and Bevan M. Baas. A complete real-time 802.11a baseband receiver implemented on an array of programmable processors. In *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, pages 165–170, Oct. 2008.

[114] Wm. A. Wulf and Sally A. Mckee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23:20–24, 1995.

[115] S. McKee and Sally A. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*, pages 162–167, New York, NY, USA, 2004.

[116] O. Sattari. Fast fourier transforms on a distributed digital signal processor. Master's thesis, University of California, Davis, Davis, CA, USA, 2004.

[117] Zhiyi Yu. *High Performance and Energy Efficient Multi-core Systems for DSP Applications*. PhD thesis, University of California, Davis, CA, USA, October 2007. `http://www.ece.ucdavis.edu/vcl/pubs/theses/2007-5`.

[118] Michael Meeuwsen, Zhiyi Yu, and Bevan M. Baas. A shared memory module for asynchronous arrays of processors. *EURASIP Journal on Embedded Systems*, 2007:Article ID 86273, 13 pages, 2007.

[119] Stephen T. Le. A fine grained many-core h.264 video encoder. Master's thesis, University of California, Davis, CA, USA, March 2010. `http://www.ece.ucdavis.edu/vcl/pubs/theses/2010-03`.

[120] Z. Xiao, S. Le, and B. M. Baas. A fine-grained parallel implementation of a H.264/AVC encoder on a 167-processor computational platform. In *IEEE Asilomar Conference on Signals, Systems and Computers*, Nov. 2011.

[121] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, and C. Kozyrakis. A case for intelligent RAM. *IEEE Micro*, 17(2):34–44, March-April 1997.

[122] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. A. Horowitz. Smart Memories: a modular reconfigurable architecture. In *International Symposium on Computer Architecture (ISCA)*, pages 161–171, June 2000.

[123] Yi Kang, Wei Huang, Seung-Moon Yoo, D. Keen, Zhenzhou Ge, V. Lam, P. Pattnaik, and J. Torrellas. FlexRAM: toward an advanced intelligent memory system. In *International Conference on omputer Design (ICCD '99)*, pages 192–201, 1999.

[124] Jung-Yup Kang, S. Gupta, and J.-L. Gaudiot. An efficient data-distribution mechanism in a Processor-In-Memory (PIM) architecture applied to motion estimation. *IEEE Transactions on Computers*, 57(3):375–388, march 2008.

[125] Firoozshahian Amin, Solomatnikov Alex, Shacham Ofer, Asgar Zain, Richardson Stephen, Kozyrakis Christos, and Horowitz Mark. A memory system design framework: creating smart memories. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 406–417, 2009.

[126] K. Mai, R. Ho, E. Alon, D. Liu, Y. Kim, D. Patil, and M. A. Horowitz. Architecture and circuit techniques for a 1.1-GHz 16-kb reconfigurable memory in 0.18-$\mu$m CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 40(1):261–275, January 2005.

[127] Shyamkumar Thoziyoor, Jung Ho Ahn, Matteo Monchiero, Jay B. Brockman, and Norman P. Jouppi. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 51–62, Washington, DC, USA, 2008.

[128] R. Mahmud. Techiniques to make clock switching glitch free. [online] Available: `http://www.eetimes.com`.

[129] Chris J. Myers. *Asynchronous Circuit Design*. John Wiley & Sons, Inc., 2001.

[130] A.T. Tran, D.N. Truong, and B.M. Baas. A low-cost high-speed source-synchronous interconnection technique for GALS chip multiprocessors. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 996–999, May. 2009.

[131] Michael J. Meeuwsen. A shared memory module for an asynchronous array of simple processors. Master's thesis, University of California, Davis, CA, USA, April 2005. `http://http://www.ece.ucdavis.edu/cerl/techreports/2005-2/`.

[132] R. Banakar, S. Steinke, Bosik Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: a design alternative for cache on-chip memory in embedded systems. In *Symposium on Hardware/Software Codesign*, pages 73–38, May 2002.

[133] B. Flachs, S. Asano, S. H. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano. A streaming processing unit for a CELL processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2005.

[134] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, f. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, March-April 2002.

[135] S. Bell, B. Edwards, et al. TILE64 processor: A 64-core SoC with mesh interconnect. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 88–89, February 2008.

[136] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, May 1993.

[137] ChipRuud Haring. The blue Gene/Q compute chip. In *HotChips 23*, Aug. 2011.

[138] Gabriel H. Loh. 3d-stacked memory architectures for multi-core processors. *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 36(3):453–464, June 2008.

[139] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile Sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 3(1):29–41, Jan. 2008.

[140] Dae Hyun Kim, K. Athikulwongse, M. Healy, M. Hossain, Moongon Jung, I. Khorosh, G. Kumar, Young-Joon Lee, D. Lewis, Tzu-Wei Lin, Chang Liu, S. Panth, M. Pathak, Minzhen Ren, Guanhao Shen, Taigon Song, Dong Hyuk Woo, Xin Zhao, Joungho Kim, Ho Choi, G. Loh, Hsien-Hsin Lee, and Sung Kyu Lim. 3D-MAPS: 3D massively parallel processor with stacked memory. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 188–189, Feburary 2012.

[141] David Fick, Ronald G. Dreslinski, Bharan Giridhar, Gyouho Kim, Sangwon Seo, Sudhir Satpathy Matthew Fojtik, Yoonmyung Lee, Daeyeon Kim, Nurrachman Liu, Michael Wieckowski, Gregory Chen, Trevor Mudge, Dennis Sylvester, and David Blaauw. Centip3De: A 3930 DMIPS/W configurable near-threshold 3d stacked system with 64 arm cortex-m3 cores. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 190–191, Feburary 2012.