Space Applications on Tilera

Justin Richardson, Chris Massie, Dr. Herman Lam, Kunal Gosrani, and Dr. Alan George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
University of Florida
Email: {richardson, massie, hlam, gosrani, george}@chrec.org

I. Introduction

This extended abstract summarizes our presentation for the *"Multicore Processors For Space - Opportunities and Challenges"* workshop, at the IEEE Space Mission Challenges for Information Technology (SMC-IT) 2009 conference. The presentation begins with an overview of the NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida and its research activities in reconfigurable computing (RC) and RC applications for space. In particular, we will focus on space applications being implemented on the Tilera TILE64 processor. These applications include:

1) A case study of a Sum of Absolute Differences (SAD) algorithm for a comparative analysis of the vector-based operations provided by the TILE64
2) A steganography application for the TILE64 processor, highlighting alternative parallelization strategies
3) A Hyper-spectral imaging (HSI) application to compare the TILE64's shared memory and DMA operations with respect to memory homing

II. Introduction to CHREC

The Center for High-Performance Reconfigurable Computing (CHREC, http://chrec.org/) is an NSF Industry/University Cooperative Research Center (I/UCRC) comprising of more than 30 leading organizations in this field from the academic, industry, and government sectors with synergistic interests and goals in adaptive and reconfigurable computing for a broad range of missions, from satellites to supercomputers. The university sites serve as the research base (faculty, students, staff) for the Center. Currently, we have four university sites: University of Florida (lead university), Brigham Young University, George Washington University, and Virginia Tech.

At the University of Florida, the faculty and students are working on a wide range of projects, many of which are applicable to space applications. The on-going projects for this year (FY2009) are summarized as follows:

A. Projects

- F1-09: System-Level Formulation and Design
  This project focuses on methods for improving developer productivity to overcome challenges and limitations inherent in the current methods of system-level application development on RC systems for space and on the ground.

- F2-09: Translation and Execution Productivity
  The goal of this project is to improve translation and execution productivity for FPGA-based applications via three major improvements to the FPGA design flow: performance analysis techniques for automatic analysis, rapid placement & routing (PAR), and in-circuit debugging.

- F4-09: Virtual Architecture and Design Automation for Partial Reconfiguration
  This project focuses on methods to ease the use of partial reconfiguration (PR) of FPGAs through the development of a Virtual Architecture for Partially Reconfigurable Embedded Systems (VAPRES) and a suite of software tools for PR design automation.

- F5-09: RC Device Architecture Exploration
  The goal of this project is to develop a fundamental research foundation to explore RC and other multicore devices and their relationships to applications. Toward that end, we have been focusing upon metrics such as computational performance, power consumption, and memory limitations of different RC and multicore devices. Furthermore, application and kernel benchmarks have been developed and evaluated on several devices to provide a concrete device comparison and validation of performance projections. In particular, three space applications that are being implemented on the Tilera TILE64 processor will be the focus of this presentation.

- F6-09: Reconfigurable & Hybrid Fault Tolerance
  This project investigates a variety of reconfigurable fault tolerance (RFT) and hybrid fault tolerance (HFT) techniques to enhance COTS-based, fault-tolerant system architectures for system-level fault tolerance for space as well as other high-performance embedded computing (HPEC) systems.

Also, other projects that we have undertaken in recent years that are related to RC and multicore applications for space include:

- F3-08: Case Studies in Application Design
  This project focused upon research, design, and analysis of FPGA-based applications, with an emphasis

upon design concepts, algorithm/architecture mapping, platform usability, and lessons learned, in terms of optimal performance, scalability, and power consumption. Several of the applications under study in this project related closely to space-based computing, such as PDF estimation for machine learning, advanced LIDAR processing, Kalman-based target tracking, multichannel communications, Hyper-spectral imaging, image enhancement, and n-body simulations.

- DM: The NASA Dependable Multiprocessor
  This project focused upon research and development of reconfigurable computing to achieve the first deployable system technology for supercomputing in space. Fault tolerance in DM can adapt to environmental conditions, with an array of disparate and flexible modes, including SIFT along with a variety of modes operating underneath, such as ABFT, TMR, SCP, CP/RB, FEMPI, etc., many available in both spatial or temporal form. RadHard technology is featured only in the management structure of DM, whereas a diverse set of powerful COTS technologies (PPC, AltiVec, FPGA) is featured for all data processing on the critical path. Application-level techniques were examined with a range of applications including LU decomposition, 2D-FFT, synthetic aperture radar, and Hyper-spectral imaging. The DM project is sponsored by New Millennium Program at NASA, Honeywell is prime contractor, and the University of Florida is lead on R&D.

For this presentation, we will focus on three space applications being implemented on the Tilera TILE64 processor, each highlighting different aspects of the capabilities of this device: an application based on a Sum of Absolute Differences (SAD) algorithm, a steganography application, and a Hyper-spectral imaging (HSI) application.

III. Benchmarks

A. SAD - Algorithm for Sum of Absolute Difference

SAD is an image processing kernel that is used to identify sections of images that have changed or moved. It is used in target detection, classification, and tracking applications among other possible applications. The SAD algorithm involves the calculation of the difference between pixels in the comparison image and input image and the sum of these differences compared to a threshold for classification. This algorithm consists of additions and subtractions but no multiplication or division operations and can be run in parallel on multiple tiles using a data-decomposition-based parallelization scheme.

This application in our suite was used to compare the different vector operations provided by the Tilera hypervisor as part of the TILE64 architecture. The vector operations that were used included a vector addition, vector subtraction, and combined SAD vector instruction. An extensive study was performed to analyze the varying performance of each of these instructions and their impact when differing numbers of tiles were used.

1024x1024 Image Size	
Serial version (S)	67,815,647 cycles
Vectorized (Sv)	56,226,324 cycles

Tiles	4	8
Parallel version (P)	17,179,810	8,796,274
Vectorized (Pv)	14,667,995	7,576,286
Speedup (P / Pv)	1.17	1.16
Speedup (S / P)	3.95	7.71
Speedup (S / Pv)	4.62	8.95
Overall Speedup (Sv / Pv)	3.83	7.42

Fig. 1. Sampled SAD Results

The SAD algorithm is used primarily for block matching. Block matching is widely used in various fields ranging from robotics to reconnaissance. Essentially, SAD is a correlation-based method in which a sample image (or a section of an image) is compared to another image, to determine a similarity criterion. The addition of SAD to our existing benchmark suite will provide results for a multi-pass, compute-intensive, pixel-level algorithm. For a base-line comparison, we have also developed the algorithm on a Freescale PowerPC (MPC7447).

In our implementation, we focused on the Tilera development with experiments determining the optimum window size/shape and search algorithms. We also experimented with various tile configurations (# of tiles used vs. performance). After applying optimizations we tabulated speedups based on MSamples / sec; a sample is defined as a pixel operation. For each pixel and its neighbors, using a 3x3 window, SAD requires nine additions, nine subtractions and nine absolute value operations.

Optimization on the Tilera platform involved using built-in vector operations and compiler optimizations. From experimentation we observed that using more than one built-in vector operation actually caused slow-downs (we hypothesize that this was because compiler optimization was no longer as effective). Parallelization was achieved using data / block decomposition of data. We used three image sizes; 64x64, 512x512 and 1024x1024. In addition, we developed a serial version *(S)*, a vectorized serial version *(Sv)*, parallel version *(P)*, and vectorized parallel version *(Pv)*. Overall speedup denoted the ratio of best serial version to that of best parallel version. Results are shown in the tables below. Only results for 4 and 8 tiles are shown, as these were where optimal speedups were achieved (in relation to utilization of tiles). A summary of the result is shown in Figure 1. The result is based on an input reference image shown in Figure 2. The corresponding sampled SAD output from a slightly shifted input is shown in Figure 3.

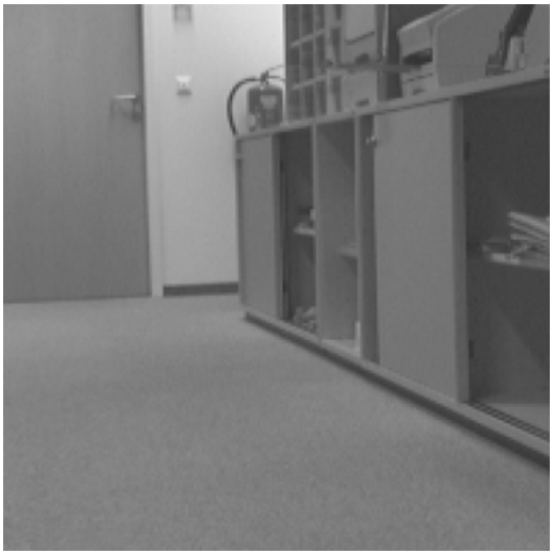


Fig. 2. Original SAD Reference Image

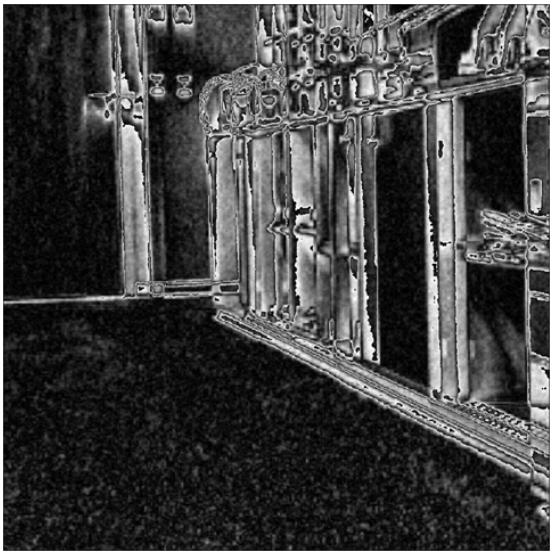


Fig. 3. Sampled SAD Output

B. Steganography

Steganography is a technique used to hide secret information in some other data without leaving any apparent evidence of data alteration. Encryption can be used to obscure the meaning of a message, but it does not obscure the fact that a message is there. Steganography hides the very presence of secret data in the carrier. The Least Significant Bit (LSB) steganography method replaces the least significant bit of a pixel in a cover image with a bit of the hidden data that the user wishes to embed (in our case, another image). The Bit Plane Complexity Segmentation (BPCS) method of steganography uses the idea that the higher order bits of a cover image can be used for embedding hidden data, provided that they are *"complex"* enough. A complexity measure is calculated to differentiate segments of an image from those that are important to the human visual system to those that are noise-like. In our implementation, the BPCS method was chosen as a benchmark tool for the purposes of our analysis of the TILE64.

Two different parallelization schemes were used in our analysis; block decomposition and a hybrid design that combines data decomposition with pipelined decomposition. In the block decomposition parallelization method, the cover image pixel space is broken into 8x8 blocks and assigned evenly in round-robin order to each of the worker nodes. In addition to these worker nodes, a leader node works to distribute the images appropriately and gather them upon completion, acting as an intermediate layer between the I/O of files and the processing being done on the worker nodes. Several optimizations were made to the design that provided insight for techniques to optimize applications on the TILE64.

The first iteration of the block decomposition program had a best-case execution time of 170 ms on TILE64, with roughly 50 ms being taken for each of the processes of distributing the cover image, processing the hidden requests, and gathering the stego image. A major inefficiency of the blocking method was that it was too granular and therefore required too much communication. The blocks were replaced by groups, which were structures consisting of multiple blocks. The result was an execution time of 88 ms due to a drop of about 40 ms from both the distribution of the cover image and gathering of the stego image. A further general optimization was made to the code to decrease the overall execution time to 67 ms.

Another optimization was to use TILE64 buffer channels instead of message passing functions, since buffered channels have less latency. After the switch, the total execution time dropped from 67 ms to 45 ms, which was primarily the result of the hidden request time dropping from 24 ms to about 6 ms. This dramatic reduction in time was due to the more efficient buffer channels being used for the sending of the conjugation map pieces back to the leader, which was a major bottleneck when message passing functions were being used. The buffer channels show the most benefit over message passing when the messages are small, which is why the larger transfers for the cover and stego images were not that much improved.

The optimal group size and number of nodes were determined on the TILE64. Figure 4 shows the total execution times as a function of group size and total number of tiles. The optimal group size is 10 blocks/group, which is potentially the case due to extra communication for smaller sizes and poorer workload balancing for larger sizes. The optimal number of tiles was 8, which is likely caused by this being the optimal balance between work distribution and communication overhead.

The hybrid design consists of two main stages, a complexity stage and an embedding stage. Originally, this design followed the methodologies of a pipeline more closely. Instead of allowing the embedding nodes to read from the cover image and perform their own Gray code conversion and bit plane slicing, the embedding nodes received this information from the complexity nodes exclusively. This method proved to be

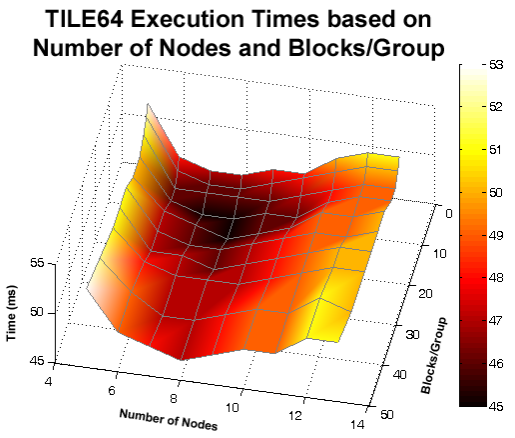


Fig. 4. TILE64 execution time

very inefficient by creating a bottleneck between the two main stages while this information was transmitted. For an example using eight tiles, sending the cover image and Gray code data for an 1152 x 768 cover image takes 25 ms. Forcing the embedding nodes to read in their own cover image negates the communication time and adds only 0.45 ms from reading from the file along with a negligible addition of time to perform the Gray code conversion before splitting the image into bit planes.

C. HSI - Hyper-Spectral Imaging

One of many key features available on the TILE64 processor is shared memory. Shared memory allows tiles across the grid to access memory also being used by other tiles. This feature allows for easier coding algorithm development. It is not, however, without its drawbacks. Shared memory that can be read from and written to by multiple processors has issues when it comes to the order of writes. Coherency is needed and in many cases it is left to the programmer to make sure that the program does not improperly read or write when another process is not completely done with that memory object.

The TILE64 processor has several functions that help the developer handle these issues, including memory fences and global barrier syncs. Shared memory also has performance implications when used on the TILE64 chip. The shared memory system on the TILE64 processor introduces a concept called *homing*. When memory that is designated to be shared is allocated, using the `malloc_shared()` command for example, whatever tile calls the allocation function becomes the *home* tile for that shared memory block. This tile then controls the accesses to this block of memory. Other tiles that want to access this shared memory block must go through this *home* tile. This has a performance impact upon an application because it can lead to a memory bottleneck and possibly congestion on the inter-tile networks. For this reason, tiles that are close together can expect to see much shorter access times to shared memory on tiles near them than to tiles that are a greater distance across the tile grid. To explore this issue the first and third stages of a Hyper-spectral Imaging algorithm were developed using two different memory layout structures. The first and third stages yield themselves well to data decomposition based parallelization strategies and that was the bases for the parallelization techniques.

Both techniques split the Hyper-spectral data cube into patches and each patch is then assigned a group of tiles in the grid to be processed. These tiles also are assigned an additional tile to hold in memory the patch of data that is to be processed. These *memory tiles* use the DMA interfaces to acquire their patch from the origin tile (Tile 0) and then use shared memory to allow the assigned processing tiles access to the patch data. Once the processing is complete the origin tile collects all the memory blocks from the memory tiles.

The first parallelization strategy that will be discussed in the presentation is the centralized memory layout. In this layout, the memory tiles are all close to the origin tile along the top row of the grid. This makes access to the origin tile faster, but the processing tiles encounter congestion during the computation. The second strategy distributes the memory tiles throughout the grid, making sure that the processing tiles are as close to their associated memory tiles as possible. This distributed method makes access to the origin tile take longer, but the processing tiles can access the local patch data faster.

The overall performance of the second method is better than the first for this HSI application because the high computational loads of the first and third stage means that most of the communications are to the local memory tiles and therefore these being closer allows for less communication overhead. The presentation will include this analysis based on qualitative analysis of preliminary data in addition to code examples (ie. microbenchmarks) of DMA transfers and the processes of shared memory allocation and their use in a TILE64 program.

IV. Conclusion

The fast pace of innovations in the field of computer architecture and software has pushed the industry into the multi-core technologies. These platforms, such as the Tilera TILE64, have many new features and challenges. This presentation highlights how these platforms can be applied to applications in the space and HPC sectors. In addition there are many techniques that developers can use to improve the performance of their applications. We have discussed data decomposition, special vector instructions, static buffered channels, shared memory, and algorithm mapping in respect to homing. These techniques and others are only a few of the available options for the TILE64 and future work will hopefully allow greater understanding of these new technologies and how users can easily take advantage of the innovations in multi-core devices for space applications.

Acknowledgment

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422.

Space Applications on Tilera

CHREC
NSF Center for High-Performance Reconfigurable Computing

2009 SMC-IT

Justin Richardson
Chris Massie
Dr. Herman Lam

Kunal Gosrani
Dr. Alan George

NSF Center for High-Performance Reconfigurable Computing (CHREC)
University of Florida

{richardson, massie, hlam, gosrani, george}@chrec.org

Outline

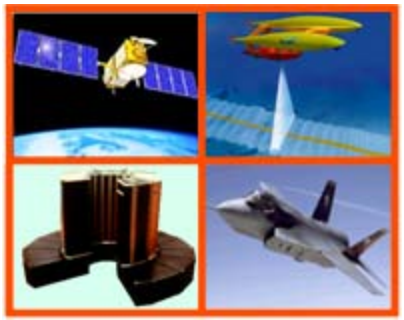
- Overview: NSF Center for High-Performance Reconfigurable Computing (CHREC)
  - CHREC mission and structure
  - CHREC research activities
- Space applications on Tilera
  - Sum of Absolute Difference (SAD)
    - Analysis of vector-based ops
  - Hyperspectal Imaging (HSI)
    - Shared memory and DMA
  - Steganography
    - Parallelization strategies
- Conclusions

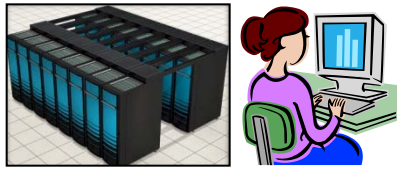
What is CHREC?

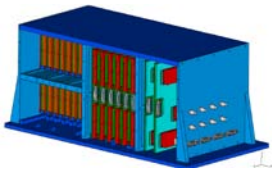
CHREC
NSF Center for High-Performance Reconfigurable Computing

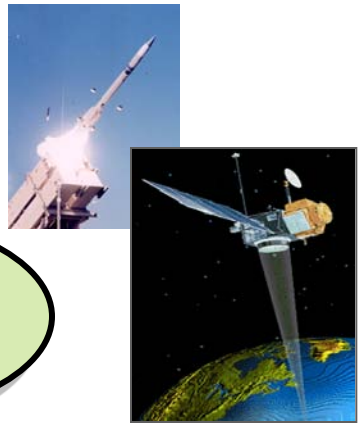
- NSF Center for High-Performance Reconfigurable Computing
  - Unique US national research center in this field, established Jan'07
  - Leading ECE/CS research groups @ four major universities
    - **University of Florida** (lead)
    - **Brigham Young University**
    - **George Washington University**
    - **Virginia Tech**
- Under auspices of I/UCRC Program at NSF
  - **Industry/University Cooperative Research Center**
    - CHREC is supported by CISE & Engineering Directorates @ NSF
  - CHREC is both a National Center and a Research Consortium
    - University groups serve as research base (faculty, students, staff)
    - Industry & government organizations are research partners, sponsors, collaborators, advisory board, & technology-transfer recipients

CHREC Mission









RC

CHREC

HPC

HPEC

Mission

Basic and applied R&D to advance S&T in advanced computing in these 3 increasingly overlapping domains.

Many common challenges, technologies, & benefits, in terms of performance, power, adaptivity, productivity, cost, size, etc.

From device/system architectures to design concepts and tools.

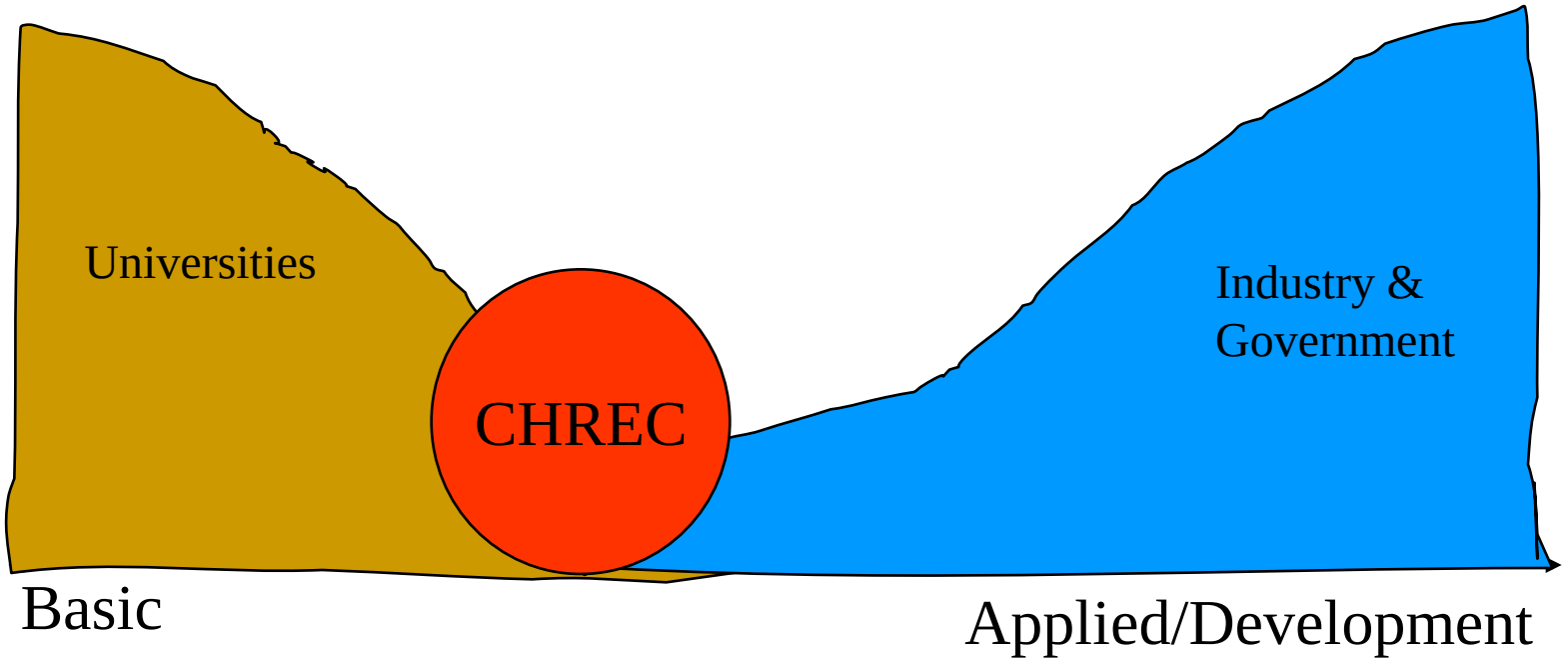
From satellites to supercomputers!

Objectives for CHREC

- Serve as foremost national research center in this field
  - Basis for long-term partnership and collaboration amongst industry, academe, and government; a national research consortium
  - RC, HPEC, HPC: from satellites to supercomputers!
- Directly support research needs of Center partners
  - Highly cost-effective manner, addressing common research interests with pooled & leveraged resources, maximized synergy
- Enhance educational experience for a large set of high-quality graduate and undergraduate students
  - Ideal recruits for ORNL (interns, engineers, researchers)
- Advance knowledge & technologies in this field
  - Commercial relevance ensured with rapid technology transfer
- Earn recognition as one of top NSF Centers
  - *Update: only 2 years young, CHREC is recognized by NSF as such!*

NSF Model for I/UCRC Centers

Research Interaction



CHREC Members

1. AFRL Munitions Directorate
2. AFRL Space Vehicles Directorate
3. Altera
4. AMD
5. Arctic Region Supercomputing Center
6. Boeing Phantom Works
7. Harris
8. Hewlett-Packard
9. Honeywell
10. Intel
11. L-3 Communications
12. Lockheed Martin MFC
13. Lockheed Martin SSC
14. Los Alamos National Laboratory
15. Luna Innovations
16. NASA Dryden Flight Research Center
17. NASA Goddard Space Flight Center
18. NASA Marshall Space Flight Center *
19. National Instruments
20. National Reconnaissance Office
21. National Security Agency *
22. Oak Ridge National Laboratory
23. Office of Naval Research
24. Raytheon *
25. Rincon Research Corp.
26. Sandia National Laboratory NM
27. SEAKR Engineering
28. Xilinx

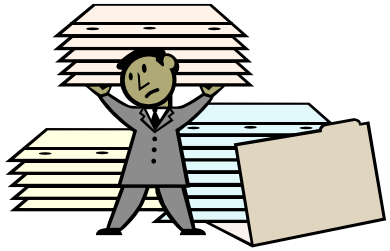
CHREC
NSF Center for High-Performance Reconfigurable Computing

Note: Many members have multiple '09 memberships (e.g. AFRL Munitions, ARSC, Boeing, Honeywell, LANL, NRO, ORNL, SNL)

* Funds pending

UF UNIVERSITY of FLORIDA
Virginia Tech
BYU BRIGHAM YOUNG UNIVERSITY
THE GEORGE WASHINGTON UNIVERSITY

CHREC Research History



- 8 projects completed in 2007 (2 schools)
  - 8 conference & journal papers approved, published
  - Variety of other results (e.g. tools, codes, cores, graduates)
  - All motivated by interests & tech transfer of CHREC partners
  - ~20 students supported
- 14 projects completed in 2008 (4 schools)
  - 17 conference & journal papers approved, published
  - Variety of other results (e.g. tools, codes, cores, graduates)
  - All motivated by interests & tech transfer of CHREC partners
  - ~40 students supported
- 12 projects underway in 2009 (4 schools)

CHREC 2008 Projects

Fault Tolerance

- **Reconfigurable Fault Tolerance and Partial RTR (F4)**
- **High-Reliability Design Tools & Techniques (B3)**
- **Reliable RC DSP/Comm Sys (B4)**

Device Studies

- **Device Characterization (F5)**
- **Heterogeneous Architectures for HPEC RC (B2)**
- **Partial RTR for HPRC (G7)**
- **Process-to-Core Mapping for Adv. Architectures (V2)**

Productivity Concepts

- **System-Level Formulation (F1)**
- **Intelligent Deployment of IP Cores (G6)**
- **Model-Based Engineering Framework (V1)**
- **Runtime Perf. Analysis (F2)**

Productivity Studies

- **Case Studies in Multi-FPGA App Design (F3)**
- **Core Library Framework (B1) Library Portability for HLL Acceleration Cores (G5)**

(where **F**=Florida, **B**=BYU, **G**=GWU, **V**=VaTech)

CHREC 2009 Projects

Architectures

- RC Device Architecture Exploration (F5)
- Characterizing and Optimizing Emerging Devices (V1)
- An API for Autonomous Adaptive Systems (V3)

Productivity Tools

- System-level Formulation and Design (F1)
- Translation and Execution Productivity (F2)
- Reuse Tools for RC Design (B1)
- Unified Parallel Programming of Tilera using UPC (G8)

Fault Tolerance & Partial Reconfiguration

- Reconfigurable & Hybrid Fault Tolerance (F6)
- Reliability Techniques for DSP/Comm Systems (B5a)
- Virtual Architecture & Design Automation for Partial Recon (F4)
- Reliable Architectures for RC (B5b)
- Virtualizing FPGA Resources for HPRCs (G7)

F5 Highlights 2009

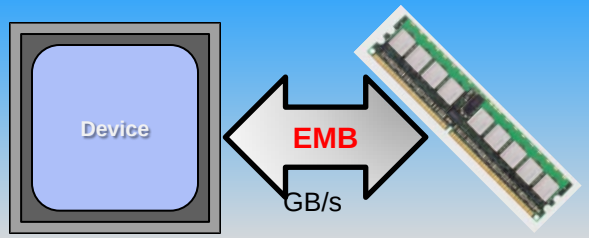
Metrics

CD/W

- New Devices
  - TigerSHARC DSP
  - TI OMAP DSP
  - Blue Gene/P
  - PACT XPP-3c
  - EP4SE680 FPGA

EMB

- Achievable memory bandwidth to off-chip memory
- Trends in available technologies



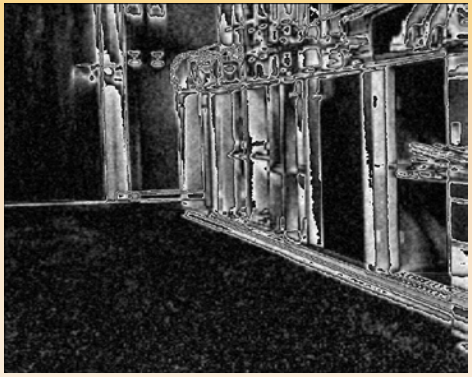
Productivity Metric Framework

- Categories
  - Learning curve
  - Code examples
  - Documentation quality
  - Programming model
  - Performance analysis methods

Benchmarks

SAD

- One data-decomposition based parallelization strategy
- Point-to-point reference matching



Hyperspectal Imaging

- Two data-decomposition based parallelization strategies
- Memory tile layout effects on performance

ACSM	• **Integer-based image processing program** • **Largest computation load in autocorrelation**
Weight Computation	• **Creates a set of weights for target detection** • **Least amenable stage for data decomposition**
Target Detection	• **Data decomposition identical to ACSM** • **Outputs detection matrix**

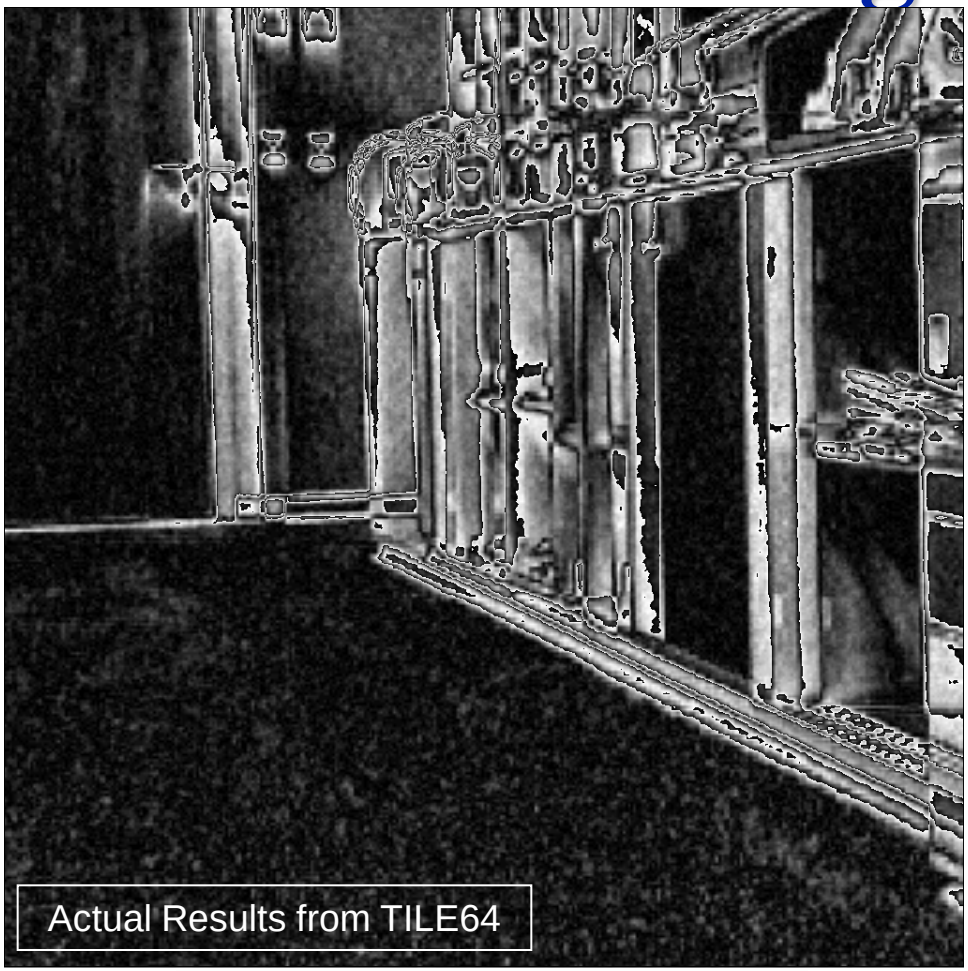
Steganography

- Two data-decomposition based parallelization strategies
- Hybrid and block approaches

SAD: Sum of Absolute Difference

- SAD
  - Used to compare two sets of similar images to estimate range of moving “targets”
  - Highest correlation between stationary objects; therefore non-stationary objects can be estimated / tracked
  - Used in applications such as target tracking and in motion estimation (robotics)
- For each pixel and its neighbors, using a 3x3 window
  - SAD: 9 additions, 9 subtractions and 9 absolute value operations
- Focus more on Tilera development
  - Determine optimum window size/shape and search algorithm
  - Determine best configuration (# of tiles used vs. performance)
  - Re-investigate window-size effect on best configuration

Example – 360 x 360 Image

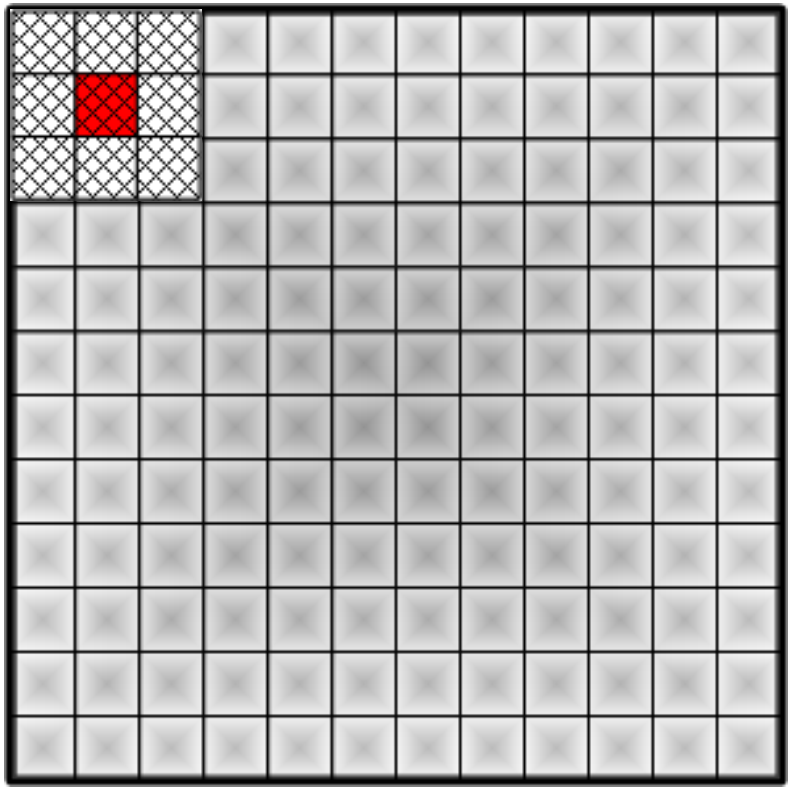


Dark regions delineate areas of no change and bright regions highlight most movement.

Resulting Correlation Map

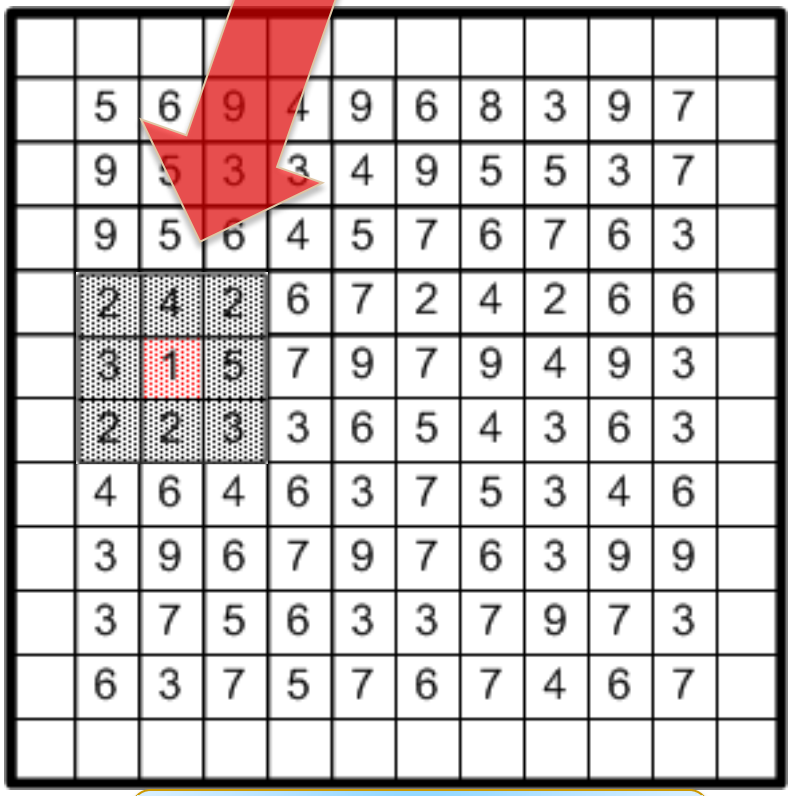
SAD Algorithm

- Motion estimation



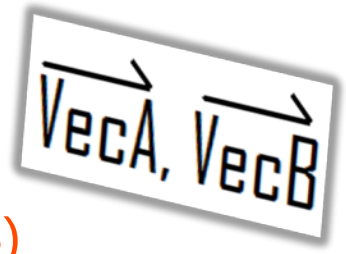
Reference Image

Highest Correlation
(Can be multiple instances)



Scanning…..

TILE64 SIMD Vector Types



- **Some useful instructions:**
  - vec_byte4 vec_addb (vec_byte4 SrcA, vec_byte4 SrcB)
    - Add four bytes in 1st source operand to four bytes in 2nd source operand ; has 1-cycle latency
  - uint32_t vec_sadb_u (vec_byte4 SrcA, vec_byte4 SrcB)
    - Sum absolute differences between four bytes in 1st source operand and four bytes in 2nd source operand; has 2-cycle latency
- **Many other instructions (provided in documentation).**
- **Following slides show results with:**
  - No optimization vs. optimized with –O2 and –O3 flags
  - Optimized with vector instructions (with –O2 and –O3)
    - vec_addb (1 vector operation)
    - Vec_addb + vec_sadb_u (2 vector operations)

SAD - Scalar Code Results

- Using Image explicitly declared (16 x 16 matrix).
  - Compiler Optimizations
    - No optimization – 5,679 cycles
    - Normal (-O2) – 796 cycles
    - Speed over size (-O3) – 735 cycles

```
sad = 0;
sad += abs( Image [x-1][y-1] - Ref [x-1][y-1] );
sad += abs( Image [x][y-1] - Ref [x][y-1] );
sad += abs( Image [x+1][y-1] - Ref [x+1][y-1] );
sad += abs( Image [x-1][y] - Ref [x-1][y] );
sad += abs( Image [x][y] - Ref [x][y] );
sad += abs( Image [x+1][y] - Ref [x+1][y] );
sad += abs( Image [x-1][y+1] - Ref [x-1][y+1] );
sad += abs( Image [x][y+1] - Ref [x][y+1] );
sad += abs( Image [x+1][y+1] - Ref [x+1][y+1] );
Final [x-1][y-1] = sad;
```

Scalar Code

SAD – Vectorized Code Results

- Using Image explicitly declared (16 x 16 matrix)
  - Vector (vec_sadb_u) + compiler optimizations
    - No optimization – 5,362 cycles
    - Normal (-O2) – 821 cycles
    - Speed over size (-O3) – 622 cycles

```
sad = 0;
sad += vec_sadb_u( Image [x-1][y-1], Ref [x-1][y-1] ) ;
sad += vec_sadb_u( Image [x][y-1], Ref [x][y-1] ) ;
sad += vec_sadb_u( Image [x+1][y-1], Ref [x+1][y-1] ) ;
sad += vec_sadb_u( Image [x-1][y], Ref [x-1][y] ) ;
sad += vec_sadb_u( Image [x][y], Ref [x][y] ) ;
sad += vec_sadb_u( Image [x+1][y], Ref [x+1][y] );
sad += vec_sadb_u( Image [x-1][y+1], Ref [x-1][y+1] );
sad += vec_sadb_u( Image [x][y+1], Ref [x][y+1] );
sad += vec_sadb_u( Image [x+1][y+1], Ref [x+1][y+1] );
Final [x-1][y-1] = sad;
```

Vectorized Code

SAD – 2x Vectorized Code Results

- Using Image explicitly declared (16 x 16 matrix)
  - Vector (vec_sadb_u, vec_addb) + compiler optimizations
    - No optimization – 5,422 cycles
    - Normal (-O2) – 823 cycles
    - Speed over size (-O3) – 637 cycles

```
sad = 0;
sad = vec_addb( sad, vec_sadb_u( Image [x-1][y-1], Ref [x-1][y-1] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x][y-1], Ref [x][y-1] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x+1][y-1], Ref [x+1][y-1] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x-1][y], Ref [x-1][y] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x][y], Ref [x][y] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x+1][y], Ref [x+1][y] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x-1][y+1], Ref [x-1][y+1] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x][y+1], Ref [x][y+1] ) );
sad = vec_addb( sad, vec_sadb_u( Image [x+1][y+1], Ref [x+1][y+1] ) );
Final [x-1][y-1] = sad;
```

Using 2 vector operations

Summary – Vector Optimization

16 x 16 Matrix		
Optimization flag	Vector + compiler Optimizations	2 Vector + compiler optimizations
None	1.06	1.05
O2 (Normal)	1.01	1.01
O3 (Speed over size)	1.18	1.15

360 x 360 Image		
Optimization flag	Vector + compiler Optimizations	2 Vector + compiler optimizations
None	1.06	1.06
O2 (Normal)	1.07	1.05
O3 (Speed over size)	1.05	1.02

- On average, Vector optimization performs better than compiler optimization
  - For both sets of data, single-vector operation performs better than using 2-vector operations
  - For small sizes, speed over size gives a better optimization whereas for larger sizes it is comparable
- Therefore single-vector optimizations with –O3 flag used for remaining results shown

Assumptions for Multi-tile Parallelization

- Image sizes chosen to be multiples of 8
- Boundary conditions
  - Instead of padding with zero's, boundary pixels ignored
    - Effective image sizes: 62x62, 510x510, and 1022x1022
- No communication overhead
  - Initial set-up time and final image-writing time not accounted for in results
- Data decomposition: Unbalanced loads exist
  - Data split into blocks depending on # of tiles specified
    - Tile 0 used for debugging + Last tile gets left-over data (animation in later slide)
    - Reported cycles for Longest Running Tile (LRT)
- # of tiles used for parallelization are multiples of two
  - From previous loop-unrolling study, non-powers of 2 performed worse
- Window size used: only 3x3.
  - Increasing window size increases change in movement (computation)
  - From paper studies, 3x3 window size is most common

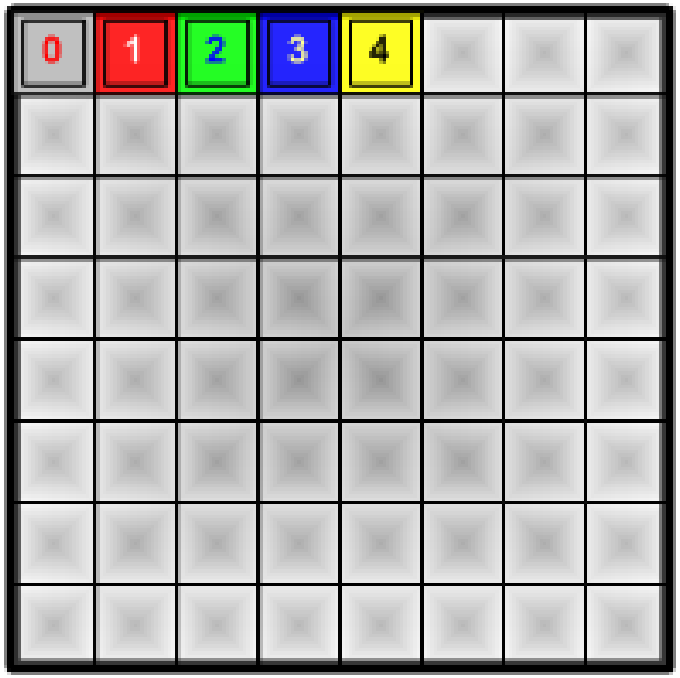
Serial Vs. Parallel

- Comparison of different serial and parallel implementations using different sizes of images and # of tiles
  - All implemented using –O3 (speed over size) optimization
  - Images sizes are 512x512 and 1024x1024
  - Serial version (S)
  - Vectorized-serial version (Sv)
  - Parallel version (P)
  - Vectorized-parallel version (Pv)
    - Overall speedup denotes ratio of best serial version to that of best parallel version
    - Speedups provided for P vs. Pv, P vs. S, Pv vs. S and Pv vs. Sv

Data decomposition

of Data Blocks = 4

Total # of Tiles = 5



8x8 Tilera Grid

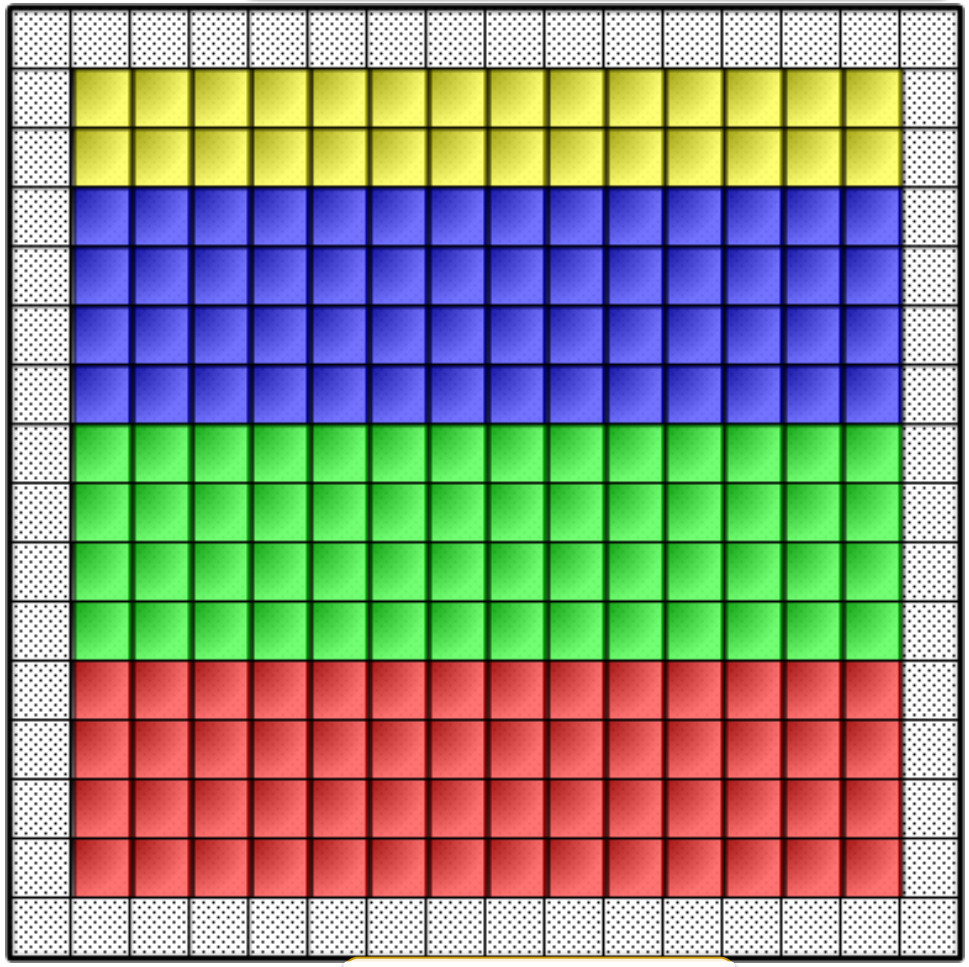


Image Pixels

Tile 0 is designated as debug tile, does no computation.

SAD: Actual Image Results

512x512 Image Size	
Serial version (S)	16,723,285 cycles
Vectorized (Sv)	13,900,769 cycles

LRT cycles

Tiles	2	4	8	16	32
Parallel version (P)	8,441,647	4,551,490	2,410,671	1,432,777	1,096,701
Vectorized (Pv)	7,192,194	3,790,997	2,090,474	1,285,146	1,078,493
Speedup (P / Pv)	1.17	1.20	1.15	1.11	1.02
Speedup (S / P)	1.98	3.67	6.94	11.67	15.25
Speedup (S / Pv)	2.33	4.41	8.00	13.01	15.51
Overall Speedup (Sv / Pv)	1.93	3.67	6.65	10.82	12.89

1024x1024 Image Size	
Serial version (S)	67,815,647 cycles
Vectorized (Sv)	56,226,324 cycles

LRT cycles

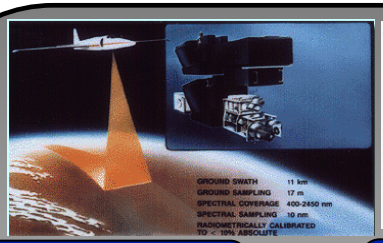
Tiles	2	4	8	16	32
Parallel version (P)	33,971,349	17,179,810	8,796,274	4,634,445	2,727,127
Vectorized (Pv)	28,925,766	14,667,995	7,576,286	4,002,613	2,459,945
Speedup (P / Pv)	1.17	1.17	1.16	1.16	1.11
Speedup (S / P)	2.00	3.95	7.71	14.63	24.87
Speedup (S / Pv)	2.34	4.62	8.95	16.94	27.57
Overall Speedup (Sv / Pv)	1.94	3.83	7.42	14.05	22.86

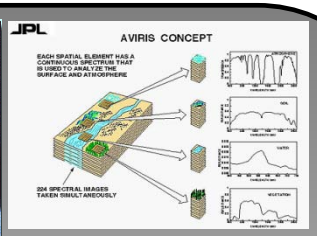
Vectorization provides higher speedups when there is enough data to churn through.

The more data each tile has to work with, the better vectorization performs (as seen in the larger 1024 x 1024 image results).

Increasing non-linear speedup with increasing # of tiles. 4 and 8 tiles give best return on speedup vs. utilization.

Hyperspectral Imaging





Input data cube of pixel vectors

Stages 1 & 3 have been studied due to their computational load

ACSM
- **Integer-based image processing program**
- **Largest computation load in autocorrelation**

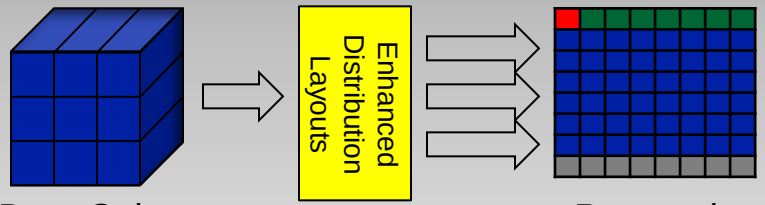
Weight Computation
- **Creates a set of weights for target detection**
- **Least amenable stage for data decomposition**

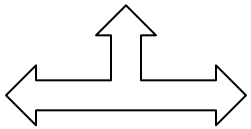
Target Detection
- **Data decomposition identical to ACSM**
- **Outputs detection matrix**

Output detection matrix

Images courtesy of:
Jet Propulsion Laboratory
California Institute of Technology

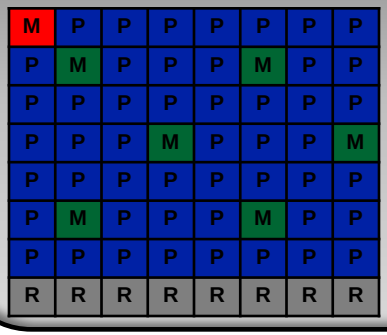
- **Hyperspectral imaging data flow**
  - **Data-decomposition (Stage 1 & 3 are pixel-independent)**
  - **Patch-based processing per tile**
  - **DMA and tile shared-memory structures (Homing)**





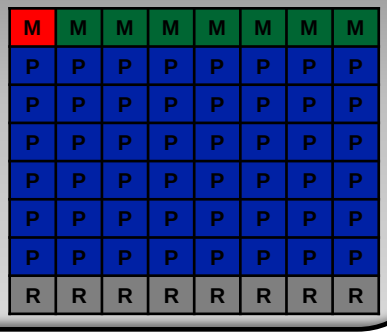
Distributed

Distributed memory tiles allows localized tile-memory interaction



Centralized

Centralized memory layout allows faster DMA transfers



Red: Master memory tiles
Green: Memory tiles
Blue: Data processing tiles
Gray: Reserved or unused

TILE64 Memory Sharing

Program Subparts

- Shared memory
  - Pointer Definition
  - Allocation
  - Homing
  - Pointer Synchronization
  - Tile Access
- DMA
  - DMA channel request
  - DMA wait

Code Examples

```
//Core Memory Allocation
size_t shared_size = N * L * sizeof( int );
if((HSIdata = (int *) malloc_shared(shared_size)) == 0){
    ilib_die("Not enough Memory");}

//Allocate other Memory
DoneHome = (int *) malloc_shared(sizeof( int ) );
*DoneHome = 6;

if((HSIacsm0 = (int *) malloc_shared(L*L*sizeof(int))) == 0){
    ilib_die("Not enough Memory");}
for(int i = 0; i < L*L; i++){HSIacsm0[i] = 0;}

if((HSIout0 = (int *) malloc_shared(N*M*sizeof(int))) == 0){
    ilib_die("Not enough Memory");}
for(int i = 0; i < N*M; i++){HSIout0[i] = 0;}

//Fence the memory
ilib_mem_fence();
```

```
//DMA Results
if(rank == 9){if(ilib_mem_start_dma(
    &HSIout0[N*M/nTilesMem*(0)],    //Dest.
    HSIout1,                        //Src.
    sizeof(HSIout1)*N*M/nTilesMem,  //Size
    &request) < 0){                 //Requ.
        ilib_die("Failed DMA2");}

        if(ilib_wait(&request, &status) < 0){
            ilib_die("Failed on DMA Wait");
        }
}
```

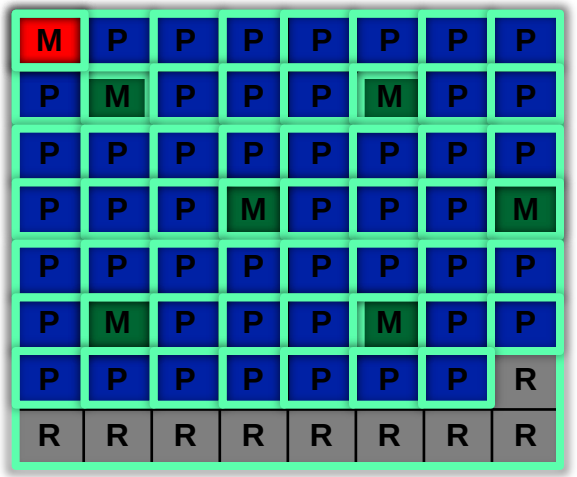
Hyperspectral Imaging Layout

Program Subparts

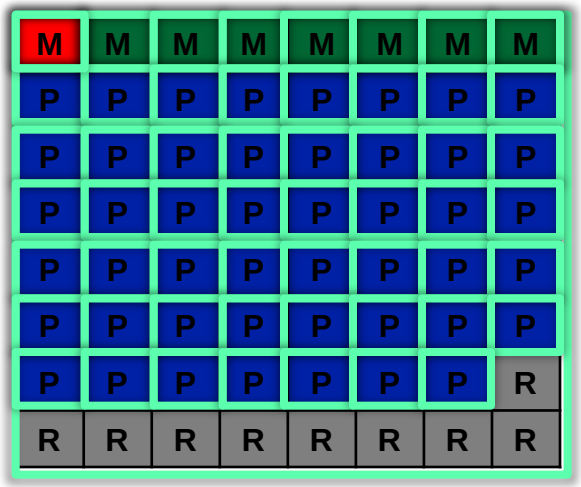
- Shared memory
  - Pointer Definition
  - Main Allocation
  - Pointer Synchronization
    - Point-to-Point
    - Broadcast
  - Tile Access
- DMA
  - DMA channel creation and request

Point-to-Point Synchronization does not require all tiles

Distributed



Centralized



HSI: Results

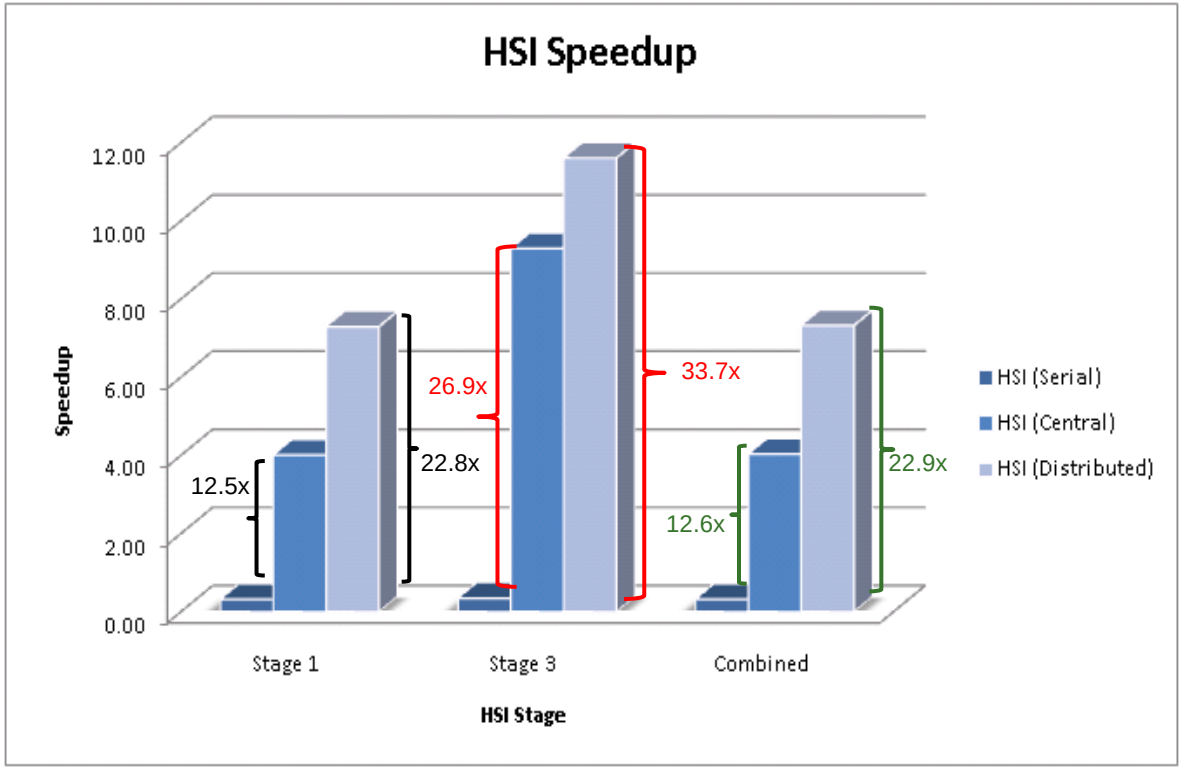
Average HSI Results									
Execution Times (seconds)					Speedup vs. PPC			Speedup vs. Serial Tile64	
Time (s)	PPC (Baseline)	Tile64 (Serial)	Tile64 (Centralized)	Tile64 (Distributed)	Speedup (Serial)	Speedup (Centralized)	Speedup (Distributed)	Speedup (Centralized)	Speedup (Distributed)
Stage 1	69.01	216.02	17.22	9.46	**0.32**	**4.01**	**7.29**	**12.54**	**22.82**
Stage 3	69.01	2.19	0.08	0.06	**0.34**	**9.29**	**11.60**	**26.93**	**33.66**
Combined	69.01	218.20	17.31	9.53	**0.32**	**4.03**	**7.32**	**12.61**	**22.90**

HSI Parameters

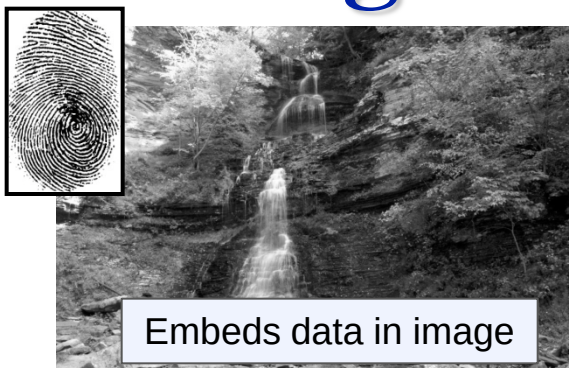
- Image Size: 48,000 pixels
- Spectral Layers: 225
- Process to Memory Distance (Max, Min)
  - Serial (0,0)
  - Centralized (12,1)
  - Distributed (3,1)

Key Points

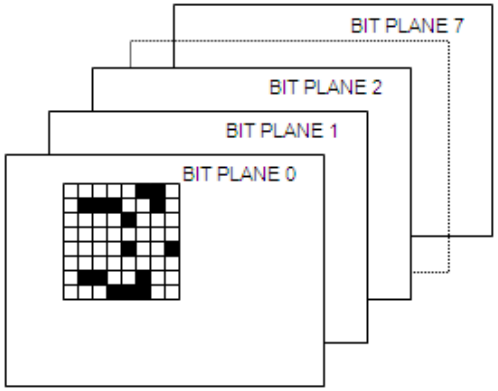
- DMA used for memory tile to master communications
- Shared memory used for memory tile to processing tile communications
- Combined speedup is limited due to computational dominance of Stage 1

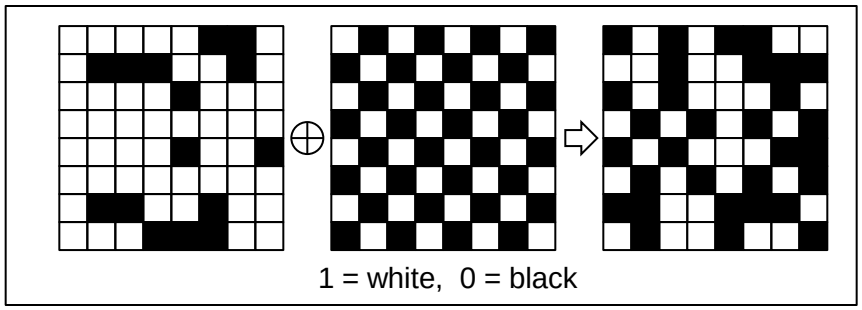


BPCS Steganography



- Gray code
- Bit plane slicing
- Complexity calc
  - α = Σ xor
- Conjugation
  - α < 45



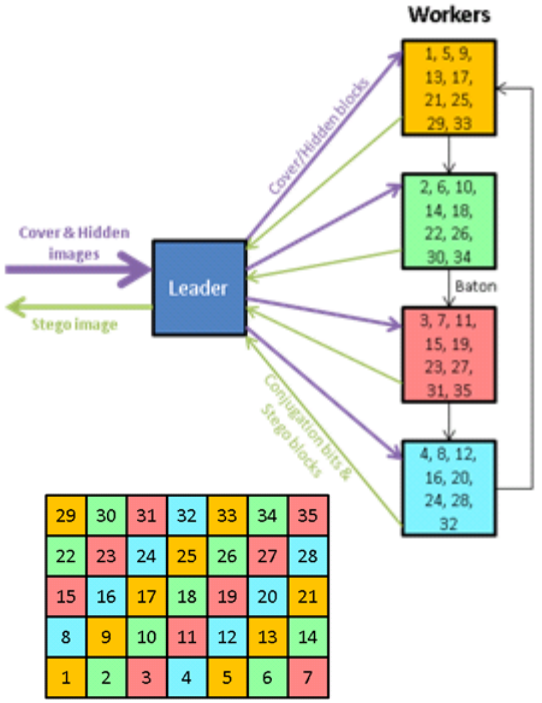


Bit-level and communication-intensive image processing app

- Two methods for parallelization:
- Block Decomposition
  - Splits data to be processed among multiple worker nodes
  - Single leader node moderates
    - Uses ilib_proc_spawn() for groups
  - Baton is passed in cyclic pattern between worker nodes to determine num of complex regions
  - Returns to leader if block was conjugated
- Hybrid Design
  - Combines pipelined and data decomposition
  - Complexity and Embedding stages
    - Half of processors for each stage
    - Uses ilib_proc_go_parallel()
    - Data decomp based on rank
  - Last node to embed becomes conjugation leader node

Block Decomp. – Packet Optimization

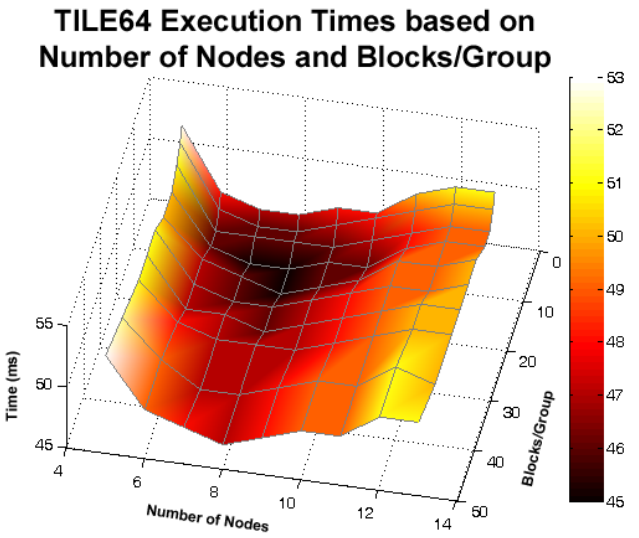
- First design iteration of block decomposition best-case execution time of 170 ms
  - Roughly 50 ms each taken for distributing cover image to workers, processing hidden image requests, and gathering output image
- Blocks too small, too much communication, not enough computation
  - 1 block = 76 byte packets
  - 10 blocks/grouping = 652 byte packets
  - 170 ms → 88 ms execution after change
  - Drop of about 40 ms from both distribution of cover image and gathering of output image



Block Decomp. – Buffered Channels

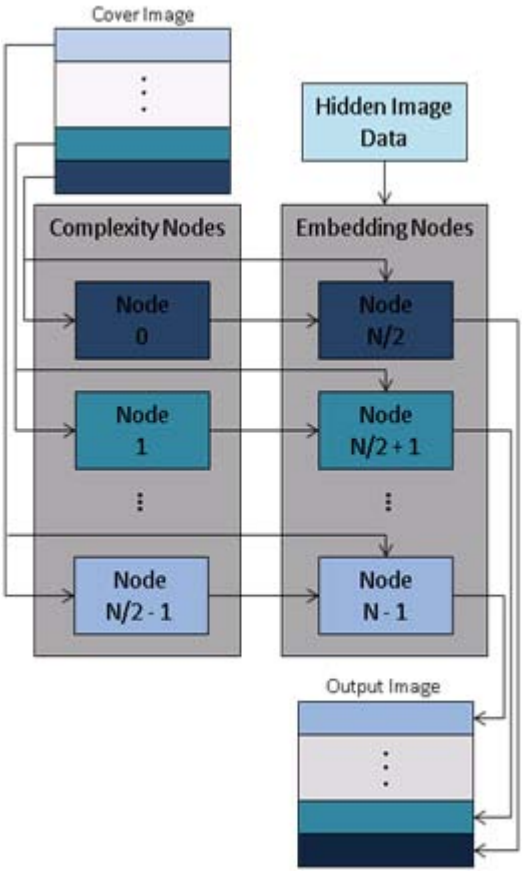
- Design uses point-to-point message passing
- Buffered channels provide socket-like interface
  - Better than message passing for repeated transmissions
  - Send or receive requires ~50 cycles

- 67 ms → 45 ms execution switching to non-blocking buffered channels
  - Sending conjugation map pieces to leader tile much more efficient
  - Hidden request time from 24 ms → 6 ms
    - Was major bottleneck with message passing
  - Buffered channels shows most benefit when messages are small
    - Larger transfers of cover and output images were not improved much



Hybrid – Repeated Computation

- Hybrid design followed pipeline design closely
- Embedding nodes received Gray code converted and bit-plane sliced data from complexity nodes
  - Very inefficient; bottleneck between two stages when data transmitted
- For 8 tiles & 1152x768 cover image:
  - Takes 25 ms to receive data
  - When embedding nodes read in data from file instead, only 0.45 ms
- Gray code conversion & bit-plane slicing resulted in low computational intensity
  - Repeated calculations can provide performance increase when $t_{communication} > t_{computation}$

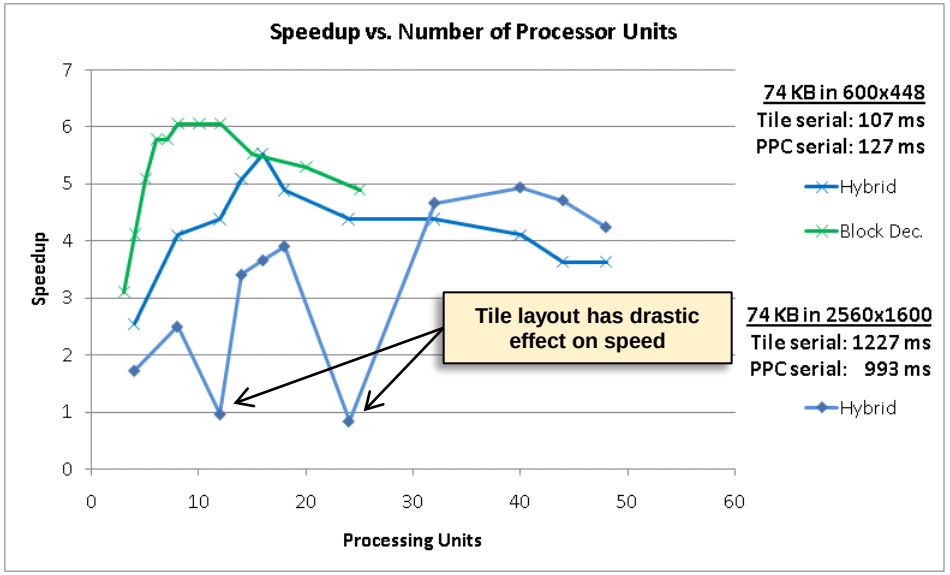


BPCS: Results

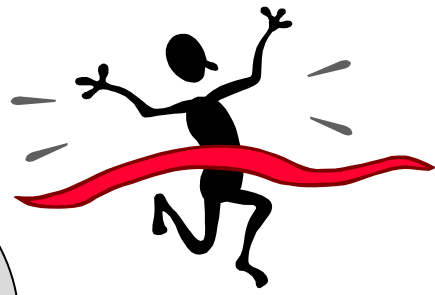
- Similar performance between TILE serial code and PPC baseline
- Block Decomposition uses Buffered Channels; Hybrid uses Message Passing
- Extrapolated best-case speedup = 24.2x
  - Replicate design 4 times on device

- When cover image size increases 15x, Hybrid design optimal with 40 tiles
  - TILE64 allows app to scale to larger # of tiles
  - Due to efficient inter-chip communication network and low latency
- Not as computationally intensive as some apps
  - Bottlenecked from heavy communication in algorithm and memory allocation

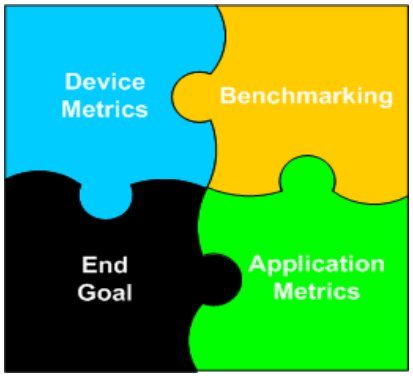
- Round robin scheme used for tile placement for Hybrid design
  - Inefficient arrangement of tile positions has drastic effect
- Odd continuation of trend in graph for Hybrid design
  - Only certain tile configurations and number cause drastic decrease in performance
  - Possible topic of future study



Conclusions



- **Benchmarks**
  - **SAD**
    - Correlation-based kernel for image processing and target tracking
    - One parallelization strategy with a max speedup of 19.2x
    - The more data each tile has to work with, the better vectorization performs
  - **HSI**
    - Image processing application for target detection and recognition
    - Examples of TILE64 shared memory and DMA interfaces
    - Data “Homing” used to help map data to processing tiles
    - Two data-decomposition-based parallelization strategies highlighting effects of data location on performance
  - **BPCS Steganography**
    - Bit-op intensive application for hiding data in other digital information
    - Finding balance for packet sizes to prevent communication saturation
    - Buffered channels show large benefit, especially when frequent, small messages are sent
    - Important to keep in mind general parallelization strategies to maximize performance



Questions?