Application Accelerators: Dues ex machina?

CCGSC, Flat Rock, North Carolina

Jeffrey S. Vetter

Oak Ridge National Laboratory
and
Georgia Institute of Technology
Background and motivation

- Current trends in architectures favor two strategies
  - Homogenous multicore
  - Application accelerators

Correct architecture for an application can provide astounding results

Challenges to adopting application accelerators

- Performance prediction
- Productive software systems

Solutions from Siskiyou

- Modeling assertions
- Multi-paradigm procedure call
The Drama

➤ Years of prosperity
  – Increasing large-scale parallelism
  – Increasing number of transistors
  – Increasing clock speed
  – Stable programming models and languages

➤ Notable constraints force a new utility function for architectures
  – Signaling
  – Power
  – Heat / thermal envelope
  – Packaging
  – Memory, I/O, interconnect latency and bandwidth
  – Instruction level parallelism
  – Market trends favor ‘good enough’ computing – Economist
Current Approaches to Continue Improving Performance

- **Chip Multiprocessors**
  - Homogenous multicore
  - Intel
  - AMD
  - IBM

- **Application accelerators to augment general purpose multi-cores**
Results from Initial Multicores Provide Performance Boost
Quad Kilo-core chips are on the way!

- 4 core chips coming
- 8 core chips likely
- ??

- **Rapport**
  - Rapport currently offers a 256 core chip
  - Planning 1024 core chip in 2007 – Kilocore™
  - Targeted at mobile and other consumer applications
Enter Application Accelerators

> Optional hardware installed to accelerate applications beyond the performance of the general purpose processor

<table>
<thead>
<tr>
<th></th>
<th>Intel Woodcrest Dual Core</th>
<th>NVIDIA Quadro FX 4500 GPU</th>
<th>NVIDIA GeForce 6600 GPU</th>
<th>IBM Cell Processor</th>
<th>ClearSpeed Avalon</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock frequency</td>
<td>3.0 GHz</td>
<td>470 MHz</td>
<td>350 MHz</td>
<td>3.2 GHz</td>
<td>250 MHz</td>
</tr>
<tr>
<td>type</td>
<td>CPU</td>
<td>accelerator card</td>
<td>accelerator card</td>
<td>CPU</td>
<td>accelerator card</td>
</tr>
<tr>
<td>power usage</td>
<td>80 W</td>
<td>110 W</td>
<td>30 W</td>
<td>100 W</td>
<td>20 W</td>
</tr>
<tr>
<td>speed single / double precision</td>
<td>~48 GFLOPS / ~24 GFLOPS</td>
<td>180 GFLOPS / NA</td>
<td>20 GFLOPS / NA</td>
<td>256 GFLOPS / 25 GFLOPS</td>
<td>50 GFLOPS / 50 GFLOPS</td>
</tr>
<tr>
<td>typical size</td>
<td>CPU socket</td>
<td>PCIe / MXM¹ card</td>
<td>PCIe / MXM¹ card</td>
<td>CPU socket</td>
<td>PCI-X card</td>
</tr>
<tr>
<td>cooling</td>
<td>heatsink + fan</td>
<td>heatsink + fan</td>
<td>HS-only or HS+fan</td>
<td>heatsink + fan</td>
<td>HS-only</td>
</tr>
</tbody>
</table>

¹ MXM: Mezzanine Memory Module
For Example ... Graphics Cards
For Example ... STI Cell
For Example ... ClearSpeed
For Example ... FPGAs
AMD Torrenza Ecosystem

Torrenza: Enabling partners to build the most exciting solutions in history

Network Processing
- Established $8 billion market in network platform
- Likely migration to server platform

Content

Security

IPTV

Processing

Transcoding

Media
- Highly competitive market in flux
- Known growth opportunities

Enablement

Enterprise Technologies
- Identified data center opportunities

Horizontal technology to open markets

Gaming Physics Accelerator

XML Accelerator

PCI-E Bridge

FLOPs Accelerator

I/O Hub

USB

PCI

8 GB/S

8 GB/S

8 GB/S
Architectures that Match Application Requirements can offer Impressive/Astounding Performance Benefits

- Geo-registration on GPU
  - 700x speedup over commodity processor
- Numerous FPGA results on integer, logic, flop applications
  - 40x on Smith-Waterman
  - 10x speedup on MD
- HPCC RandomAccess on Cray X1E
  - 7 GUPS on 512 MSPs
  - 32 GUPS on 64,000 procs

### Molecular Dynamics

<table>
<thead>
<tr>
<th>System</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell PPE</td>
<td>0.425</td>
</tr>
<tr>
<td>MTA2 w/32 procs</td>
<td>~0.035</td>
</tr>
<tr>
<td>2.2GHz Opteron</td>
<td>0.125</td>
</tr>
<tr>
<td>Cell w/ 8 SPEs</td>
<td>0.013</td>
</tr>
<tr>
<td>GPU (7900GT)</td>
<td>0.012</td>
</tr>
</tbody>
</table>
Disruptive Technologies and the S-Curve

Déjà vu?
- Floating Point Systems accelerator (1970-80s)
- Weitek coprocessors (1980s)

Some differences …
- Flops are free
- Power and thermal envelopes are constraining designs
Significant Hurdles to Adoption for Accelerators (and multicores?)

- **Performance prediction**
  - Should my organization purchase an accelerator?
  - What will be the performance improvement on my application workload with the accelerator?
  - Is the accelerator working as we expect?
  - How can I optimize my application for the accelerator?

- **Productive software systems**
  - Do I have to rewrite my application for each accelerator?
  - How stable is the performance across systems?
Performance Modeling
Modeling Assertions Introduction

- We need new application performance modeling techniques for HPC to tackle scale and architectural diversity
  - Performance modeling is quite useful at many stages in the architecture and application development process

- Existing approaches
  - Manual
    - Application driven
  - Automated
    - Target architecture driven
  - Black box schemes—accurate but applicability to a range of applications and systems is unknown

- Goals
  - Aim to combine analytical and empirical schemes
  - A framework for systematic model development – performance engineering of applications
  - Modular
  - Hierarchical
  - Separate application and system variables
  - Based on ‘user’ or ‘code developer’ input—no magical solution
  - Scalable—future application and system configurations
Symbolic Performance Models with MA

Modeling Assertion (MA) = Empirical data + Symbolic modeling

- Advantages over traditional modeling techniques
  - Modularity, portability and extensibility
  - Parameterized, symbolic models are evaluated with Matlab and Octave
- Construct, validate, and project application requirements as a function of input parameters

Declare important application variables

Declare important application operations

Annotate code with MA API

Validate Modeling Assertions empirically at runtime

Terminate when model is representative & error level is acceptable

Incrementally refine model based on error rates by adding and modifying variable and operation declarations
MA Framework

MA API in C (for Fortran & C applications With MPI)

Classes of API calls currently implemented and tested

ma(f)_subroutine_start/end
ma(f)_loop_start/end
ma(f)_flop_start/stop
ma(f)_heap/stack_memory
ma(f)_mpi_xxxx
ma(f)_set/unset_tracing

Source code annotation

Runtime system generate trace files

Model validation

Control flow model

Symbolic model

Post-processing toolset (in Java)

main ()
{
    ....
    loop (NAME = conj_loop) (COUNT = niter)
    {
        loop (NAME = norm_loop) (COUNT = l2npcols)
        {
            mpi_irecv (NAME = nrecv) (SIZE = dp * 2);
            mpi_send (NAME = nsend) (SIZE = dp * 2);

            send = niter*(l2npcols*(dp*2)+l2npcols*(dp)+
cgitmax*(l2npcols*(dp*na/num_proc_cols)+dp*na/num_proc_cols+l2npcols*(dp)+l2npcols*(dp)+l2npcols*(dp))
        }
    }
}
Example with MA Annotation

call maf_def_variable_int('na', na)
call maf_def_variable_int('nonzer', nonzer)
   ....
call maf_def_variable_assign_int('num_proc_cols',
   > '2^ceil(log(nprocs)/(2*log(2)))', num_proc_cols)
   ....
call maf_loop_start('conj_loop', 'niter', niter)
do it = 1, niter
   ....
call maf_flop_start('flopzeta', '4*na/num_proc_cols',
   > 4*na/num_proc_cols)
do j = 1, lastcol-firstcol+1
   norm_temp1(1) = norm_temp1(1) + x(j)*z(j)
   norm_temp1(2) = norm_temp1(2) + z(j)*z(j)
   enddo
   call maf_flop_stop('flopzeta')
   ....
call maf_loop_end('conj_loop', it-1)
   ....
call maf_subroutine_start('conj_grad')
   ....
call maf_mpi_irecv('l2rcv', 'dp*na/num_proc_cols',
   > dp*naa/npcols, l2npcols)
call maf_mpi_irecv('l2rcv', 'dp*na/num_proc_cols',
   > dp*naa/npcols, l2npcols)
call mpi_irecv( q(reduce_recv_starts(i)),
   > reduce_recv_lengths(i),
   > dp_type,
   > ....
call maf_subroutine_end('conj_grad')

Input parameters: na, nonzer, niter and nprocs

Derived parameters: nz, num_proc_cols, l2cpcols and dp (size of REAL)

A loop with loop count

For floating-point operation count

End markers used for validation

Markup for subroutine invocation

MA MPI API call
Example Model Validation

**NAS CG**
- Class S: na=1400, nonzer=7
- Class W: na=7000, nonzer=8
- Class A: na=14000, nonzer=11
- Class B: na=75000, nonzer=13
- Class C: na=150000, nonzer=15

**NAS SP**
- Class S: problem_size=7
- Class W: problem_size=36
- Class A: problem_size=64
- Class B: problem_size=102
- Class C: problem_size=162

```
opq: ma_flop:7000:7000:0.0: PASS=50: FAIL=0
cj_sumred: ma_loop:1:1:0.0: PASS=50: FAIL=0
l4rcv: ma_mpi_irrecv:8:8:0.0: PASS=50: FAIL=0
l4snd: ma_mpi_isend:8:8:0.0: PASS=50: FAIL=0
sumred: ma_flop:1:1:0.0: PASS=50: FAIL=0
floprrhopq: ma_flop:21001:21001:0.0: PASS=50: FAIL=0
cj_rho: ma_loop:1:1:0.0: PASS=50: FAIL=0
l5rcv: ma_mpi_irrecv:8:8:0.0: PASS=50: FAIL=0
l5snd: ma_mpi_isend:8:8:0.0: PASS=50: FAIL=0
flopnzx: ma_flop_start:3503:4347:-0.194: PASS=0: FAIL=2
```
Computation Distribution

- Runtime distribution across loop blocks in NAS SP and CG
  - Generated using symbolic models
  - Vary important parameters, such as number of processors, apps parameters
- Unlike CG, there is not a single hotspot in SP
MPI Message Distribution Analysis

- **CG**
  - 65% messages in CG are 8 bytes
  - Remaining over 37 Kbytes

- **SP**
  - 95% messages in SP are ~28 Kbytes
  - Remaining 50-64 Kbytes

- **Conclusion:** CG requires low latency network

Speedup of NAS CG and SP on ORNL Cray XT3 system
Sensitivity of SP calculations

Sensitivity of workload requirements with respect to the SP input parameter: problem_size
Modeling Assertions with Accelerators

- MA framework provides critical information on computational intensity and data movement that is critical for mapping applications to accelerators.
- MA is providing insight into DOE applications for acceleration:
  - Biomolecular application: AMBER
  - Climate Modeling: POP
Mapping Amber Kernel to FPGAs

Obtained 3x application speedup on FPGA using HLL on SRC 6C MapStation.

jac Amber8 benchmark:

List time (% of nonbond) = 4.72 (5.19)
Direct Ewald time = 70.82
Recip Ewald time = 14.76
Total Ewald time (% of nonbond)= 86.23 (94.81)

FFT time (% of Recip) = 4.76 (32.24)
MPPS: Multi-Paradigm Programming System
Several vendors are designing, even now building *multi-paradigm* systems

- Along with general purpose microprocessors, a multi-paradigm system may include:
  - FPGAs
  - Highly multi-threaded processors (MTA)
  - Graphics processors
  - Physics processors
  - Digital signal processors

- Vendors include:
  - IBM, SGI, Cray, SRC, ClearSpeed, ClearSpeed, ClearSpeed, Linux Networx

*Legend*
- P: commodity processor
- V: vector processor
- GPU: graphics processing unit
- FPGA: field programmable gate array
Multi-Paradigm Computing Challenges

» Multi-Paradigm systems offer lots of performance potential, but…

» …it is challenging to realize that potential
  – Different APIs, different tools, different assumptions!
  – Different ISAs, SDKs
  – Explicit data movement
  – Simplistic scheduling
  – Static binding to available resources
MPPS Basis: Multi-Paradigm Procedure Call (MPPC)

- Multi-Paradigm Procedure Calls
  - Adopt highly successful RPC approach
  - Open protocol for communication within infrastructure

- MPPC runtime system
  - Runtime agent to manage access to device
  - Directory service for dynamic discovery of devices and their status
  - Local service OS on devices (if possible)

- Support for defining adaptive policies for scheduling application requests onto computing devices
  - Simple policies built-in
  - Custom policies can be driven by automated administration and performance tools
Compiler Support for MPPS

Pragma identify regions of code to accelerate
  – Built on Open64
  – Similar to OpenMP analysis

Extracts code for device service
  – Device code compiled separately with device specific SDK

Replaces original code with MPPC call
  – Marshals data; starts, waits on device
Summary

- **Accelerators will continue to gain market share in one form or another**
  - Expansion slots
  - On-chip accelerators which are used as necessary

- **Software systems that can mask the complexity will become much more important**
  - Multi-paradigm Programming System
  - Automated generation of MPPC calls

- **Performance modeling and analysis will become critical for procurements, validation, and optimization**
  - Modeling assertions
This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

http://www.csm.ornl.gov/ft

vetter@computer.org
Bonus Slides
Performance Stability

SPE Optimizations

Runtime (sec) [log scale]

- Original: 0.681 s
- Fast cosine: 0.248 s
- Fast exp/sqrt: 0.064 s
- SIMD: 0.047 s
Performance Stability (2)

⇒ HPC Challenge ratio of Optimized over Baseline

![Bar chart showing performance stability ratios for HPL, RA, TRANS, FFT, and STREAMS]
MPI Symbolic Models

Error rate for MPI message sizes and count = 0%

Message size (bytes) and message count per MPI task for NAS MPI CG and SP benchmarks
Sensitivity Analysis: Data Generated by Symbolic Models

- Application input parameters:
  - na (array size)
  - nonzer (number of nonzero elements)
- Question: which parameter influences the workload and how?

- MA models generated the required information efficiently
- Observation: the nonzer parameter has a huge impact on computation requirements
- Also identified that nonzer has no impact on MPI communication
MPPS Research Directions

Integration with Modeling Assertions
- MA models can help MPPC make better scheduling decisions
- MPPC behavior can be fed back to improve models that are multi-paradigm aware

Multi-operation scheduling
- Instead of MPPC_FFT, MPPC_DGEMM granularity, turn over larger sequences of work to MPPC infrastructure
- More optimization opportunities
- More scheduling burden on MPPC infrastructure
```c
int main( int argc, char* argv[] )
{
    MPI_Init( argc, argv );
    MPPC_Init();

    ...  
    MPPC_DGEMM( a, b, s, z );
    ...
    MPPC_ZDFFT( u, v, n );
    ...

    MPPC_Finalize();
    MPI_Finalize();
    return 0;
}
```

Mapping, data marshaling, scheduling of specific multi-paradigm device hidden from user.

Automated static analysis and profile-directed feedback can hide conversion of applications to MPPC and optimize series of MPPC routines.