Toward Simplifying Application Development on Heterogeneous Multicore Platforms

Barbara Chapman
University of Houston

Chattanooga, September 24, 2009
Agenda

- The Accelerator Challenge
- Directives for Accelerators?
- A Few Examples
- Possible Directions
That was then: Multicore

- Small number of cores, shared memory
- Each core: single thread or multithreaded, complex or simplified
- Resources (L2 cache, memory bandwidth): shared or separated
- System built from homogeneous or heterogeneous cores
ClearSpeed Accelerator: CSX600 designed for HPC

- **Processor Core:**
  - 40.32 64-bit GFLOPS
  - 10W typical
  - 210MHz
  - 96 PEs, 6 Kbytes each
  - 8 redundant PEs

- **SoC details:**
  - integrated DDR2 memory controller with ECC support
  - 128 Kbytes of SRAM

- **Design details:**
  - IBM 130nm process
  - 128 million transistors (47% logic, 68% memory)

- **Sampled Q3 2005**
This is Now: Accelerators Galore

- **Today’s accelerators:**
  - Mostly Nvidia GPGPUs
  - Program via CUDA

- **Two levels of parallelism:**
  - DOALL (fully parallel, outer loops)
  - Synchronous (SIMD or vector, inner loops)
    - No SIMD/Vector operations
  - Operate on rectangular domains of 2 or 3 dimensions

- **No DOALL synchronization**
  - synchronizations at inner level
  - Weak consistency model, i.e memory flushes after barriers
The Execution Model

- **Host-accelerator device model**
  - Accelerator memory limited, distinct from host memory
  - No virtual address mapping between them

- **Host offloads accelerator regions**

- **Host handles:**
  - Memory allocation on device
  - Initiate data transfer
  - Sending kernel code to device
  - Waiting for completion
  - Transfer results back from device
  - Queuing kernels for execution
GPGPU Programming Challenges

- Low-level APIs
- Requires major code change
  - Many limitations on kind of code that can be executed
  - Branching leads to high inefficiencies
  - CPU (cache) optimizations generally not useful, not good starting point for accelerator code
- Large design space for accelerator kernels
  - Need to carefully consider memory and register usage
  - Number of threads per block
  - Loop optimizations
  - Data prefetching, offloading
- Small differences in code can lead to large differences in performance
Heterogeneous Cores: A High-Level Programming Model?

- Heterogeneous (accelerator) programming is currently very low-level
- Number of questions to be resolved if we are to provide one programming model across gen.-purpose CPUs and accelerators

- How to identify code that should be accelerated?
- How to move data between host cores and other devices?
- What is role of user?
Can We Standardize?

- There is not just one kind of accelerator
  - GPGPU, FPGA, DSP, ARM, Cell, Vector
  - Different range of instructions

- Programming model for accelerators must suit variety of architectures and applications
  - What kind of code will run on accelerator? Arbitrary sequential regions, loops, parallel code?
  - How do we optimize for memory use?

- What kinds of applications need to be supported?
  - “Streams” programming?
  - Embedded applications?

MCA develops low-level interfaces to “glue” heterogeneous components
The OpenMP ARB

- OpenMP is maintained by the OpenMP Architecture Review Board (the ARB), which
  - Interprets OpenMP
  - Writes new specifications - keeps OpenMP relevant
  - Works to increase the impact of OpenMP

- Members are organizations - not individuals
  - Current members
    - Permanent: AMD, Caps Entreprise, Cray, Fujitsu, HP, IBM, Intel, Microsoft, NEC, PGI, SGI, Sun, Texas Instruments
    - Auxiliary: ASCI, cOMPunity, EPCC, KSL, NASA, RWTH Aachen
Existing OpenMP-like approaches

- **PGI**
  - Define a region of code running on accelerators

- **CAPS**
  - Define codelet running on accelerators and data transfer

- **Acotes Project (Barcelona, INRIA,..)**
  - Define tasks on accelerators

- **IBM**
  - Overload OpenMP directives, avoid extensions
What kind of code region can be mapped to accelerator?

- OpenMP parallel region
- Worksharing construct
- Tasks
- Arbitrary well-structured region
- All of the above?
CAPS: Hybrid Application View
CAPS HMPP Approach

- Preserve application code legacy
  - Integrate HWAs in applications with minimal disruption
  - Provide HWA interoperability

- Mix HMPP and OpenMP to exploit HWAs and general purpose cores
  - Use best parallel version according to execution context
  - OpenMP directives in codelets
    - Codelet will instantiate threads
  - OpenMP directives outside codelets
    - One codelet per OpenMP thread using HWA
Existing Approaches: CAPS HMPP

- Declare hardware specific implementations of functions (HMPP codelets)
  - Can be specialized to the execution context (data size, ...)

- Codelet calls (RPC)
  - Synchronous, asynchronous properties

- Data transfers
  - Data prefetching

- Synchronization barriers
  - Host CPU will wait until remote computation is complete
CAPS: Multiple Devices

- Use \#D accelerators in parallel

```c
#pragma omp parallel for, private (j)
    for (jj=0; jj<\#D; jj++){
        for (j=jj*(n/\#D); j<jj*(n/\#D)+(n/\#D); j++){
            #pragma hmpp tospeedup1 callsite
            simplefunc1(n,t1[j],t2,t3[j],alpha);
        }
    #pragma hmpp tospeedup1 release
    }
```
Existing Accelerator Approaches: PGI

- Compiler directives for to accelerate regions of C/Fortran code.
  - OpenMP-like
  - Incremental development

- Features:
  - Initializes accelerator
  - Manages data and program transfers between host and accelerator
  - Directives are hints, not commands
  - User guidance: data scoping, mapping of loops, performance details.
PGI Directives

- C:
  
  #pragma acc region [clause [,clause]...] new-line

- Fortran:
  
  !$acc region [clause [, clause]...]
Clauses

- !$acc region [clause [, clause]…]
  - if (condition)
  - copy (variable list)
    - copy in from host to accelerator
  - copyin(variable list)
    - copy in from host to accelerator
  - copyout(variable list)
    - copy out from accelerator to host
  - local(variable list)
    - Local variable in accelerator
Clauses

- $\texttt{!$acc do [clause [, clause]…]}$
  - host [(width)]
  - parallel [(width)]
  - seq [(width)]
  - vector [(width)]
  - unroll (width)
  - kernel
  - shortloop
  - private ( variable list )
  - cache ( variable list )
Example

double precision A(rowa,cola), B(cola,colm), C(rowa,colm),

!$acc region
!$acc& copy(c(1:rowa,1:colm)), copyin(b(1:cola 1:colm),a(1:rowa,1:colm))
do j=1,colm
    do i=1,rowc
        sum = 0.0d00
        do k=1,cola
            sum = sum + a(i,k) * b(k,j)
        enddo
        c(i,j) = c(i,j) + sum
    enddo
enddo
!$acc end region
Comments

- Kernels generated from loop nest in accelerated region
  - Compiler must prove loops are independent!

- Compiler will attempt to find two levels of parallelism
  - Across Multiprocessors (parallel)
  - Within Processor/Warps (vector)

- Compiler will attempt to find a good schedule
  - Vector lengths scheduled to warps
  - Strip-mine loops to achieve desired schedule
  - Define parallel loops distributed across processors
  - Determine Grid and Block sizes in GPU
    - Block size fixed by compiler, grid size will dependent on the size of the data passed to GPU
Output from compiler:

39, Generating copy(c(1:rowa,1:colb))
    Generating copyin(b(1:cola,1:colb))
    Generating copyin(a(1:rowa,1:cola))
40, Loop is parallelizable
41, Loop is parallelizable
    Accelerator kernel generated
    40, !$acc do parallel, vector(16)
    41, !$acc do parallel, vector(16)
43, Scalar last value needed after loop for sum
    Loop carried scalar dependence for sum
    Inner sequential loop scheduled on accelerator
44, Accelerator restriction: scalar variable live-out from loop: sum

Compiler generates
Block size: 16x16
Choices: The Productive Approach

High-level approach

- Less code modification, potentially portable
- Directives are easiest to fit in with status quo: Prescriptive or hints?
- Some amount of adaptivity to given configuration and workload: code should still run even if accelerator is not available
- Data management is crucial. Persistent data, timing of allocation, de-allocation and transfer

Implementation

- Does the user prescribe or influence some of major decisions?
  - Number of threads in block? Loop optimizations?
  - Data movement?
- Tools for experimenting with different alternatives?

A number of companies are exploring this intensively
OMAP35x processor: Laptop like performance at handheld power level

**Performance**
- High-performance Superscalar ARM® Cortex™-A8 featuring NEON co-processor with immersive 2D/3D Graphics accelerator
- HD video decode utilizing TMS320C64x+ DSP and video hardware accelerators
- Low power utilizing TI’s SmartReflex™ technology with option for integrated and discrete Power Management ICs

**Features**

### Cores
- Cortex A-8 with NEON™ SIMD Coprocessor / DSP-based TMS320C64x + DSP and video accelerators (max performance only)
  - 600 MHz / 430 MHz @ 1.35V (operating limits apply)
  - 550 MHz / 400 MHz @ 1.27V
- 2D/3D Graphics Engine - Up to 10M polygons per second

### Memory
- ARM:
  - 16 kB I-Cache; 16 kB D-Cache; 256kB L2
- TMS320C64x+ DSP and video accelerators
  - L1 32kB Program Cache/32kB Data Cache + 48kB SRAM
  - L2 64kB Program / Data Cache + 32 kB SRAM; 16 kB ROM
- On Chip: 64kB SRAM; 112kB ROM

### Peripheral Highlights
- Support for LPDDR1
- Support for NOR, NAND, SRAM, Pseudo SRAM
- USB 2.0 HS Compliant OTG Controller w/ 2 additional USB Host Controllers
- Display subsystem with LCD and TV interface. Supports PIP, color space conversion, resize and rotation.
- Camera I/F with CCD controller and Image-pipe (Preview, Resize, Statistics)

### Package 1 (CBB)
- 12x12 mm, 0.4mm pitch, Package On Package (POP); 515 pin PBGA; production now; can be used with discrete memory

### Package 2 (CUS)
- 16x16 mm 0.65 mm pitch, 423 pin PBGA; production now. Utilizes Via Channel™ Array Technology with 0.8mm pitch plus design rules.

### Package 3 (CBC)
- 14x14 mm, 0.5 mm pitch POP; 515 pin PBGA; production now; must use POP memory

**Applications include:**
- Automotive Infotainment
- In-dash navigation
- Consumer
  - PND
  - PMP
  - Digital Video Camera
- Medical
  - Patient monitoring
- Industrial
  - Portable ultrasound
  - Point of sale
  - Smart white goods

**OMAP35x Processor**

- **ARM® Cortex™-A8 CPU**
- **POWERVR SGX™ Graphics (3515/3530 only)**
- **C64x+™ DSP and video accelerators (3525/3530 only)**

**Display Subsystem**
- LCD Controller
- Video
- 10 bit DAC

**Camera I/F**
- Image Pipe
- Parallel I/F

**Peripherals**
- **Connectivity**
  - USB 2.0 HS OTG Controller
  - USB Host Controller x3
  - HDQ / 1-wire
  - SDRC
  - GPMC
  - MMC/SD/SDIO x3
  - Timers
  - GP x12
  - WDT x2

**Note:** Peripheral limitations may apply among different packages

POWERVR SGX™ 3D engine is licensed from Imagination Tech. Ltd.
Questions?

SCENIC VIEW OF NEXT GOAL
Evaluation of the PGI Fortran & C Accelerator Programming Model
Evaluation

- Experiments were done with 1 process and accelerator (Tesla C1060) per node.
- Application Bugget: Fox and Leyman parallel matrix algorithms with different algorithms and matrix sizes per node.
- Compiler: PGI 9.0.3
- Machine: lens.ccs.ornl.gov
  - 32 node cluster
  - Nodes with 4 sockets, 64GB memory
    - quad-core AMD Opteron 8356
  - Accelerators:
    - **Tesla C1060**, 4GB memory, 30 multi-processors, 240 cores
      - 32 warp size, Capability 1.3
    - **GeForce 8800 GTX**, 800MB memory, 16 multi-processors, 128 cores
      - 32 warp size, Capability 1.0
Matrix size 1000x1000

Matrix Multiplication Kernel Time

GPU Kernel Time (Grid: 63x63, Block: 16x16)

- Data Transfer Time
- Kernel Computation Time
- Initialization
- Total Kernel Time

Ratio Host Only vs. GPU Kernel Time

Improvement Factor (x)
Observations

- Two levels of parallelism generated
  - Grid size 63x63, Block size 16x16

- Host version of Matrix-Multiplication not optimized

- $O(n^2)$ data transfer vs. $O(n^3)$ computations
  - Transfer time vs. Kernel Computation time.
  - Average of 19% of total GPU kernel time is transferring data.

- GPU version of matrix multiplication highly parallel
  - ~98x improvement

- The matrix algorithm is not optimized for the Host version. (No blocking/tiling, etc)
Different Sizes of Matrix Multiplication

Matrix Multiplication Kernel Time

Ratio Host Version/ GPU Kernel Time

GPU Kernel Time Breakdown (%)

- Data Transfer
  - 4000x4000: 4.06%
  - 2000x2000: 6.64%
  - 1000x1000: 12.86%

- Computation
  - 4000x4000: 75.58%
  - 2000x2000: 67.42%
  - 1000x1000: 95.54%

- Initialization
  - 4000x4000: 0.40%
  - 2000x2000: 2.53%
  - 1000x1000: 19.72%
Observations

- As problem size increases, host-version of matrix multiplication suffers from cache size and memory bandwidth.
  - True even for blocked/tiled best version (see later).
- Improvement ratio increases from 98x to 200x as problem size increases
- Data transfer time becomes less relevant for bigger problem sizes.
  - 96% is spent computing the kernel in GPU for 4000x4000
  - 4% in data transfer from host to GPU.
- Small kernel sizes suffer from one time overhead initialization.
do 00100 ib = 1,nbi
   ilo = (ib-1)*bsi + 1
   ihi = min((ilo+bsi-1),rowc)
do 00200 jb = 1,nbj
   jlo = (jb-1) * bsj + 1
   jhi = min((jlo+bsj-1),colc)
do 00300 kb = 1,nbk
   klo = (kb-1) * bsk + 1
   khi = min((klo+bsk-1),cola)
!$acc region
   do 00400 j = jlo,jhi
      do 00500 k = klo,khi
         myconstant = b(k,j)
      do 00600 i = ilo,ihi
         c(i,j) = c(i,j) + a(i,k)*myconstant
      00600       continue
   00500       continue
   00400       continue
!$acc end region
Another idea: “Places”

- Define a place that code and data can be associated with
- A place can then be associated with an accelerator
- Complete data environment in the place
- Define data communication via streams/pipes
- Flexibility in combination with the current OpenMP programming model
Place and Data

- Define the number of places and associate with hardware attribute
  - `omp_place_t p[N] // for static declaration`
  - `P[N-1] = omp_GPU  // the last place is GPU`

#pragma omp place(p[N-1])
  - shared – data shared among all places
  - `shared_on_place(variables) – data shared on the current place`
  - Private – data private to each task/thread