A Hardware Independent Parallel Programming Model

Magne Haveraaen, Eva Suci
University of Bergen
Department of Informatics

NWPT
10-12 October 2007, Oslo
Programming Parallel Architectures

• Multi-core earthquake (Fran Allen - PACT07)
  – GPUs, IBM's Cell, dual-core or quad-core

• Architecture-aware programming
  – Existing codes cannot be ported directly to new architectures
  – New architectures come along with new programming models: one for GPUs, one for Cell and so on.

• Hardware independent programming
  – Map computations to new architectures without rewriting the problem solving code
Massively Parallel Chips

- Graphics vendors support general-purpose computations on GPUs
  - NVIDIA's GeForce 8800 with CUDA
  - AMD's Stream Processor with CTM

- Sony's Playstation 3 with IBM's Cell BE processor
  - AMD’s R600 series
Hardware-based Programming Models

- GPU architecture vs. Cell BE architecture – alike, yet completely different
Data Dependency Algebras (DDA)

• Programs manipulate data
  – data dependency view of a program:
    • computations are driven explicitly by dependencies
    • data is propagated along the dependencies
• Data dependency algebras are powerful algebraic abstractions to describe dependencies
  – Signature of a DDA:
    • set of points
    • set of branch indices
    • two operations for traversing the dependency:
      – \( r \) request
      – \( s \) supply
  – Satisfying certain axioms
• Sapphire is a programming language based on DDAs
• The Bitonic Sort DDA is defined by:

  BS = sb int * row int * col int – the points
  branch indices \{0,1\}
  request (r) and supply (s) functions

• r defines where data is requested from
• s defines where data needs to be supplied

• The computation is defined as a recursive function on the points of the graph, using explicitly the r function (in Sapphire terminology):

  \[
  V(p) = \begin{cases} 
  \text{if } ((1 \leq \text{sb}(p) \leq h) \text{ and } (0 \leq \text{row}(p) \leq \text{sb}(p)-1)) \text{ then} \\
  \quad \text{if } (\text{bit}(\text{sb}(p),\text{col}(p)) = \text{bit}(\text{row}(p),\text{col}(p))) \\
  \quad \quad \text{then } \min(V(r(p,0)),V(r(p,1))) \\
  \quad \quad \text{else } \max(V(r(p,0)),V(r(p,1))) \\
  \quad \text{else if } ((\text{sb}(p) = 1) \text{ and } (\text{row}(p) = 1)) \text{ then } v_{\text{col}(p)} 
  \end{cases}
  \]
Hypercube space-time DDA of dimension 4 is defined by:

- **HST** = node int * time int – the sort for the points
- branch indices: \{0,1,2,3,4\}
- request and supply functions:
  
  \[
  r(p,b) = \begin{cases} 
  &\text{HST(node(p),time(p)-1)} \\
  &\text{HST(flip(b-1,node(p)),time(p)-1)} \\
  \end{cases} \\
  s(p,b) = \begin{cases} 
  &\text{HST(node(p),time(p)+1)} \\
  &\text{HST(flip(b-1,node(p)),time(p)+1)} \\
  \end{cases}
  \]
Embedding BS DDA onto HST DDA

- An embedding consists of:
  - Mapping BS points to HST points
  - Mapping request branches at a BS point into incoming communication channels in HST DDA
  - Mapping supply branches at a BS point into outgoing communication channels in HST DDA

- Having defined the embedding, the bitonic sorter can be compiled for the hypercube
Hardware 2: CUDA Space-Time DDA (CUST)

GPU program: sequence of kernels

- GPU kernel: grid of blocks
  - Threads within one block:
    - communicate through shared memory, can synchronise
  - Threads in different blocks:
    - communicate asynchronously via main GPU memory
    - not able to exchange data within the same kernel.

- The CUDA space-time DDA is defined:
  - CUST = space CUB * time int,
    CUB   = block int * thread int
  - branch indices: CUB (thread identifiers)
  - request and supply functions:
    - r(p,b) = CUST(b,time(p)-1),  s(p,b) = CUST(b,time(p)+1)
    - block(b)=block(space(p)) intra-block communication
Embedding BS DDA into CUST DDA

- BS points mapped to CUST points:
  \[ \text{EP}(p) = \text{CUST}(\text{CUB}(\text{col}(p)/n, \text{col}(p)\%n), \text{grow}(\text{sb}(p), \text{row}(p))) \]
- BS DDA request branches mapped to paths
  \[ \text{ER}(p,b) = \text{CUB}(\text{col}(\text{r}(p,b))/n, \text{col}(\text{r}(p,b))\%n) \]
- BS DDA supply branches mapped to paths
  \[ \text{ES}(p,b) = \text{CUB}(\text{col}(\text{s}(p,b))/n, \text{col}(\text{s}(p,b))\%n) \]

where \( n \) is the number of threads within a block

Colouring:

- 4 green blocks in a row ~ 1 kernel
- 1 green block ~ consists of 4 threads
- 1 grey frame across:
  1. Threads write to GPU memory, and the kernel terminates
  2. A new kernel is invoked, and starts with threads fetching data from GPU memory

BS DDA for 16 inputs, embedded into CUST DDA, executed by 4 kernels, each consisting of 4 blocks of threads.
A Hardware Independent Programming Model

Algorithm 1

DDA 1

HDDA 1

Hardware 1

Algorithm 2

DDA 2

HDDA n

Hardware n

new embedding 1

embedding 1

embedding n

new embedding n

...
Summary

• DDAs are very powerful, high level abstractions to describe:
  – data dependencies of programs
  – space-time communication of hardware architectures (e.g. Hypercube, CUDA)

• DDA-Embeddings are explicit

• It is a powerful theory at work:
  – Sapphire – prototype compiler - produces parallel code from DDAs using the MPI library
  – efficiency of different embeddings can be easily tested

• DDAs support portability of old codes onto new architectures
Future Work

• Define the space-time communication layout of other multi-core systems in terms of DDA descriptions (e.g. Cell)

• Enhance Sapphire to produce code for CUDA, Cell BE

• Getting Benchmark results of different embeddings for the same computation
  – low cost: redefine the embedding on a high level and get Sapphire to compile the code

• Jump to low-level hardware and investigate possible ways of using DDAs in circuit design.