Array Allocation in Non-Cached Memory Systems

Justin L. Tripp
and Trident Compiler Team
Los Alamos National Laboratory

October 2006
Overview

• Introduction
• Non-Cached Memory Systems
• Trident’s Approach
• Results
• Conclusions and Future Work
Scientific Applications

- Scientific Computing is typically floating point (singles and doubles)
- Scientific Computing requires support of legacy software
- Much of the software has high-level parallelism already extracted (e.g., MPI, etc.)
- It is desirable to provide acceleration through more sophisticated high-level compilers.
Coprocessor Acceleration

- Scientific Computing pushes the boundaries of computing
- Current microprocessors are not increasing in speed as they have in the past.
- Solution – acceleration of code through “commodity” co-processors
- An age old hardware solution to accelerating hard tasks. (Think 8087.)
- Examples Floating-Point co-processors, FPGAs, GPUs, Physics Processors, I/O controllers (SCSI, RAID, Ethernet, etc.)
Non-Cached Memory Systems

- Embedded Processors – 2kB to 64kB Scratch pad Memory, DRAMS, ROMS (Flash, EEPROM, etc.)
- Cell - 256KByte Local Store (LS)
- Clearspeed coprocessor - Internal 128KB SRAM, 6KByte LS, External DRAMs
- FPGAs - Internal Block Rams, External SRAMs, External DRAMs
FPGAs

Field Programmable Gate Arrays (FPGAs)

- Reprogrammable Logic
- Lookup Tables (LUTs), small RAMs, Registers, and Routing
- No *free* lunch – configurability costs
- Slower clock speeds, increased parallelism, more special blocks (PPC405, DSP blocks, embedded multipliers, etc.)
- Very successful in signal and image processing, interconnects, glue logic
- FPGAs continue to grow in size and slightly in speed.
FPGAs
FPGAs

- FPGAs are extremely Flexible.
- FPGAs provide a way to provide custom hardware.
- FPGA tools require some engineering knowledge.
- FPGA tools take a long time to run.
- FPGAs are normally programmed using VHDL or Verilog (HDL Languages).
- However, HDL Languages are very low-level and difficult
FPGAs can perform high-performance floating-point (FP) operations. However, traditional FPGA development is difficult and few tools exist to specifically aid FP hardware development.

Trident Goals:

- Accept C input with double and float data types.
- Automatically extract available parallelism.
- Automatic pipelining of loops.
- Allow the selection from different FP libraries.
- Allow user developed FP libraries.
Four principal phases of compilation:

- LLVM front-end
- Trident IR Transformation
- Scheduling
- Synthesis

**LLVM - Low Level Virtual Machine (www.llvm.org)**
**IR - Intermediate Representation**
LLVM Intermediate Representation (IR)

- RISC-like three address code ($a = \text{add float } b, c$)
- SSA Static Single Assignment form (infinite virtual register set and explicit dataflow)
- Simple low-level control flow constructs create explicit control-flow graph (ret, br, switch)
- Load/store instructions with typed pointers
- Explicit language-independent type information (void, bool, int, float, double, ushort, ..., pointer, array, structure, function)
- Explicit typed pointer arithmetic (getelementptr takes a pointer and returns element address)
Trident IR

Control Flow Graph

IR is used for:
Optimization, Resource allocation, Scheduling, Operation Mapping.

Operations also include:
• Start and Stop times
• Operand Type
• Hardware Reuse set
• Operator Class Information
Hardware Analysis and Instruction Scheduling

There is always limited memory bandwidth and physical resources on any FPGA chip. Ensuring successful implementation of the circuit on the target chip and achieving maximum execution speed requires analysis of the hardware and scheduling of the available resources.

- **Hardware Analysis**
  - Preliminary schedule to determine times of memory reads and writes
  - Array to memory allocation
  - Logic space requirements analysis

- **Instruction Scheduling (schedule type chosen by user)**
  - non-loop code - ASAP, ALAP, Force-Directed
  - loop code - Modulo scheduling
Array Allocation

How is this different than a scratch pad RAM for an embedded processor?
Array Allocation

How is this different than a scratch pad RAM for an embedded processor?

- Wider Memory access
  - More banks of SRAM – Multiple Busses – more Bandwidth!
  - Wider data sizes (32bit, 64bit)
  - Many different kinds of Memory (SRAM, BlockRAM, DRAM)
  - Any number of registers (no “stack” variables) and a small number of allocatable BlockRAMs

- No “Code” – if you want to do it, you must make it. Functional call caching does not make sense.

- DRAM requires a stall mechanism in the “code”. Pipelined sections must wait.

- DMA engines (or memory transfer engines) must be built.
Array Allocation

No memory allocation provided on FPGAs - Where should it go?

• Strategy – Using a greedy algorithm:
  ★ Schedule memory accesses to maximize memory bus usage.
  ★ Allocate data to memories.
  ★ Evaluate memory allocation on # of cycles required.
  ★ Repeat minimizing the # of cycles.

• Evaluate using function: \( c = \text{cost\_function} - \text{attempts} \)
  If \( c \leq 0 \), then schedule the memory access.
Choosing a Memory Allocation

Algorithm:

- Check to see if there is memory available
- Iterate through each memory and each array
- Check the cost in \# of cycles for this array in this memory
- Optimize through minimization in the change in the overall allocation cost
Array Allocation – Outer loop

- Pre-allocate to memories to understand read and write latencies
- Preliminary schedule operations to know where the reads and writes are
- Choose a memory allocation
- Evaluate memory allocation on # of cycles required
Cost Function

The cost function depends on the number and kind of busses or ports that a memory has. It is not limited to one kind of memory but supports:

- Single Port RAM
- ROM
- Dual Data Dual Port RAM
- Multi-Port RAM
- Single Port

![Diagram of memory types](image)
Cost Function

The cost function considers the load on a memory’s bus caused by a given memory access.

For example:

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Schedule</th>
<th>Corrected Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read A, Read A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Here due to a bus conflict, the second read must be moved to the second cycle. The cost is two since the schedule must be lengthened to accommodate the second read.

For the different memory types, the costs are different.

- ROM - 2
- SP RAM - 2
- DDP RAM - 2 (busy address)
- DP RAM - 2 (two reads)
- MP RAM - 1 (many ports)
## Scheduling

### Scheduling Example:

\[ O = (A \times B) + D \times (C + D) \]

**ASAP**

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmp1 = A*B</td>
<td>tmp3 = tmp2*D</td>
<td>O = tmp1+tmp3</td>
</tr>
<tr>
<td>tmp2 = C+D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ALAP**

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmp2 = C+D</td>
<td>tmp1 = A*B</td>
<td>O = tmp1+tmp3</td>
</tr>
<tr>
<td></td>
<td>tmp3 = tmp2*D</td>
<td></td>
</tr>
</tbody>
</table>

**Force Directed**

<table>
<thead>
<tr>
<th>Chosen location</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmp1 = A*B</td>
</tr>
<tr>
<td>tmp2 = C+D</td>
</tr>
<tr>
<td>tmp3 = tmp2*D</td>
</tr>
<tr>
<td>O = tmp1+tmp3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tmp1 = A*B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tmp2 = C+D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tmp3 = tmp2*D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O = tmp1+tmp3</td>
<td></td>
</tr>
</tbody>
</table>
## Allocation Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Allocation Options</th>
<th>Schedule Type</th>
<th>Trident Exec. Time (in sec)</th>
<th># of Hyperblocks</th>
<th>Avg Cycles per Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patmat</td>
<td>Mult. Alloc.</td>
<td>FD</td>
<td>10.043</td>
<td>5</td>
<td>7.600</td>
</tr>
<tr>
<td>Patmat</td>
<td>1 Alloc. + Presch.</td>
<td>FD</td>
<td>9.658</td>
<td>5</td>
<td>8.600</td>
</tr>
<tr>
<td>Patmat</td>
<td>1 Alloc. + No Presch</td>
<td>FD</td>
<td>6.208</td>
<td>5</td>
<td>7.800</td>
</tr>
<tr>
<td>Patmat</td>
<td>Mult. Alloc.</td>
<td>ASAP</td>
<td>6.530</td>
<td>5</td>
<td>7.400</td>
</tr>
<tr>
<td>Patmat</td>
<td>1 Alloc. + Presch.</td>
<td>ASAP</td>
<td>4.439</td>
<td>5</td>
<td>7.400</td>
</tr>
<tr>
<td>Patmat</td>
<td>1 Alloc. + No Presch</td>
<td>ASAP</td>
<td>3.601</td>
<td>5</td>
<td>7.400</td>
</tr>
<tr>
<td>Patmat</td>
<td>Mult. Alloc.</td>
<td>ALAP</td>
<td>7.042</td>
<td>5</td>
<td>6.400</td>
</tr>
<tr>
<td>Patmat</td>
<td>1 Alloc. + Presch.</td>
<td>ALAP</td>
<td>4.510</td>
<td>5</td>
<td>7.600</td>
</tr>
<tr>
<td>Patmat</td>
<td>1 Alloc. + No Presch</td>
<td>ALAP</td>
<td>3.740</td>
<td>5</td>
<td>7.600</td>
</tr>
<tr>
<td>Euclid</td>
<td>Mult. Alloc.</td>
<td>FD</td>
<td>48.050</td>
<td>1</td>
<td>86.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>1 Alloc. + Presch.</td>
<td>FD</td>
<td>29.370</td>
<td>1</td>
<td>87.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>1 Alloc. + No Presch</td>
<td>FD</td>
<td>34.383</td>
<td>1</td>
<td>87.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>Mult. Alloc.</td>
<td>ASAP</td>
<td>8.155</td>
<td>1</td>
<td>86.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>1 Alloc. + Presch.</td>
<td>ASAP</td>
<td>7.179</td>
<td>1</td>
<td>87.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>1 Alloc. + No Presch</td>
<td>ASAP</td>
<td>5.959</td>
<td>1</td>
<td>87.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>Mult. Alloc.</td>
<td>ALAP</td>
<td>8.902</td>
<td>1</td>
<td>82.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>1 Alloc. + Presch.</td>
<td>ALAP</td>
<td>6.704</td>
<td>1</td>
<td>82.000</td>
</tr>
<tr>
<td>Euclid</td>
<td>1 Alloc. + No Presch</td>
<td>ALAP</td>
<td>6.158</td>
<td>1</td>
<td>87.000</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>Mult. Alloc.</td>
<td>FD</td>
<td>41.677</td>
<td>37</td>
<td>6.432</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>1 Alloc. + Presch.</td>
<td>FD</td>
<td>70.943</td>
<td>37</td>
<td>6.568</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>1 Alloc. + No Presch</td>
<td>FD</td>
<td>39.949</td>
<td>37</td>
<td>6.432</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>Mult. Alloc.</td>
<td>ASAP</td>
<td>35.424</td>
<td>37</td>
<td>6.243</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>1 Alloc. + Presch.</td>
<td>ASAP</td>
<td>44.390</td>
<td>37</td>
<td>6.243</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>1 Alloc. + No Presch</td>
<td>ASAP</td>
<td>34.724</td>
<td>37</td>
<td>6.297</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>Mult. Alloc.</td>
<td>ALAP</td>
<td>36.320</td>
<td>37</td>
<td>6.405</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>1 Alloc. + Presch.</td>
<td>ALAP</td>
<td>42.284</td>
<td>37</td>
<td>6.405</td>
</tr>
<tr>
<td>Skeletonization</td>
<td>1 Alloc. + No Presch</td>
<td>ALAP</td>
<td>35.600</td>
<td>37</td>
<td>6.405</td>
</tr>
</tbody>
</table>
## Memory Type Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Allocation Options</th>
<th>Memory Type</th>
<th>Trident Exec. Exec (sec)</th>
<th># of Hyperblks</th>
<th>Avg Cyc. per Blk</th>
</tr>
</thead>
<tbody>
<tr>
<td>syntheticMemTest</td>
<td>Mult. Alloc.</td>
<td>DP</td>
<td>14.789</td>
<td>3</td>
<td>22.33</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>Mult. Alloc.</td>
<td>DDP</td>
<td>16.226</td>
<td>4</td>
<td>31.50</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>Mult. Alloc.</td>
<td>SP</td>
<td>18.681</td>
<td>4</td>
<td>31.50</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>1 Alloc. + Presch.</td>
<td>DP</td>
<td>17.619</td>
<td>4</td>
<td>27.00</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>1 Alloc. + Presch.</td>
<td>DDP</td>
<td>16.368</td>
<td>3</td>
<td>22.33</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>1 Alloc. + Presch.</td>
<td>SP</td>
<td>17.919</td>
<td>4</td>
<td>27.00</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>1 Alloc. + NoPresch.</td>
<td>DP</td>
<td>9.936</td>
<td>4</td>
<td>31.25</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>1 Alloc. + NoPresch.</td>
<td>DDP</td>
<td>11.477</td>
<td>4</td>
<td>27.00</td>
</tr>
<tr>
<td>syntheticMemTest</td>
<td>1 Alloc. + NoPresch.</td>
<td>SP</td>
<td>9.574</td>
<td>4</td>
<td>31.50</td>
</tr>
</tbody>
</table>

This was a synthetic test written to saturate the read and write busses. The code has 8 times more reads than writes. The Dual Ported RAM performs best with the best allocation, but other allocations the results are mixed.
Conclusions and Future Work

Results

- Best Allocation was found using multiple allocations with prescheduling
- Force Directed does move operations away, but usually results in a longer schedule.
- Complex method can take along time to execute

Future work

- Incorporate BlockRams for smaller data blocks
- Add ability to stall blocks and use DRAM
- Use profile analysis to determine SRAM, BlockRAM, or DRAM
- Evaluate the benefit of using SRAM as local cache for DRAM
Questions?

Web address:
http://trident.sourceforge.net/

E-mail:
jtripp@lanl.gov