Analysis and Optimizations for the SSS

Mattan Erez – Feb. 2002
Outline

- Architecture overview
- Software system overview
- Simple Analyses
  - the minimum to get a program to run
- Optimizations
  - across nodes (multi-node)
  - within a node (single node)
Architecture overview (1)

- **Stream Processor**
  - 64 FPUs
  - 64 GFLOPS

- **16 x DRDRAM**
  - 2 GBytes
  - 38 GBytes/s
  - 20 GBytes/s
  - 32 + 32 pairs

- **Node**
  - Node 2
  - Node 16

- **Board**
  - Board 2
  - 16 Nodes
  - 1K FPUs
  - 1 TFLOPS
  - 32 GBytes
  - 160 GBytes/s
  - 256 + 256 pairs
  - 10.5" Teradyne GbX

- **On-Board Network**

- **Board 64**
  - Cabinet 2
  - 64 Boards
  - 1K Nodes
  - 64K FPUs
  - 64 TFLOPS
  - 2 TBytes
  - 5 TBytes/s
  - 8K + 8K links
  - Ribbon Fiber

- **Intra-Cabinet Network**
  - (passive-wires only)

- **Inter-Cabinet Network**
  - All links 5 Gb/s per pair or fiber
  - All bandwidths are full duplex

- **Bisection**
  - 64 TBytes/s

---

Architecture overview (2)

Stream Processor
- Scalar Processor
- Stream Register File
- Address Generators
- Memory Control
- On-Chip Memory

Stream Execution Unit
- Cluster 0
- Cluster 1
- ... Cluster 15
- Comm
- To/From SRF (260 GB/s)

Network Interface
- Network Channels (20 GB/s)
- Local DRDRAM (38 GB/s)

On-Chip Memory

LRF BW = 1.5TB/s

Network Channels
- Network Interfaces

Memory Control

Local DRDRAM
- Scratch Pad

Software system overview

Domain-specific languages

- MC Radiation Transport
- ODEs
- PDEs

Stream Language
Collections - ordered and unordered
Map, Filter, Expand, Reduce

Stream Virtual Machine
Streams, SRF, Memory, Sync, Nodes, Net

Low-Level Language
Threads, Streams, Memory Mgt, DSP

Legacy Code

Machine Independent
Parameterized Machine Independent
Machine Dependent
Simple Analyses (1)

- Partition streams and data
  - no virtual memory
    - spread data across nodes (and make sure it fits)
  - shared memory
    - doesn’t really matter how

- Insert and honor synchronization
  - Brook semantics imply sync
    - reductions
    - memory operations – deriving streams might require synchronization with remote nodes
  - translate into system primitives
  - scalar code memory allocation and such
Simple Analyses Example

ZeroField(force);

pospairs = pos0.selfproduct();
forcepairs = force.selfproduct();

MolclInteractions (pospairs, forcepairs, &wnrg);
Insert synchronization and tree combine
MolclSpringForces (pos0, force, &sprnrg);
Insert synchronization and tree combine
VelocUpdate (force, veloc);

kinnrg=0;
KineticEnergy(veloc, &kinnrg);
Insert synchronization and tree combine
Do on a single node:
totnrg = wnrg + sprnrg + kinnrg;
Barrier (continue)

VelocUpdate (force, veloc);

PostnUpdate (veloc, pos0);

kernels
Reductions
Inserted synchronization

- Each node gets an equal share of the input data
- A copy of the stream code executes on every node (using a mini-OS)
- Sync when necessary
- Communicate through memory
Synchronization Example

```c
kernel void KineticEnergy (molclField veloc,
reduce double *kinnrg) {

double okinnrg, h1kinnrg, h2kinnrg;

okinnrg = 0.5 * OMASS * (veloc->o[0]*veloc->o[0] +
                          veloc->o[1]*veloc->o[1] +
                          veloc->o[2]*veloc->o[2]);

h1kinnrg = 0.5 * HMASS * (veloc->h1[0]*veloc->h1[0] +
                          veloc->h1[1]*veloc->h1[1] +
                          veloc->h1[2]*veloc->h1[2]);

h2kinnrg = 0.5 * HMASS * (veloc->h2[0]*veloc->h2[0] +
                          veloc->h2[1]*veloc->h2[1] +
                          veloc->h2[2]*veloc->h2[2]);

*kinnrg += okinnrg + h1kinnrg + h2kinnrg;
```

“atomic” add across all nodes:
local add on each cluster $\rightarrow$ local combine on each node $\rightarrow$
final global combining tree: (barrier$\rightarrow$accumualte$\rightarrow$)$^n$
Simple Analyses (2)

- Convert conditionals
  - Code may contain if statements
  - Hardware supports predication and conditional streams only

- VLIW scheduling
  - Simple scheduling – possibly single IPC

- Register spilling
  - LRF → Scratchpad
  - Handle scratchpad overflows

- Double buffering
  - double buffer all streams - trivial SRF allocation
  - Kernels execute to completion and spill/reload values from memory
kernel void MolclInteractions (waterMoleculePair pstn,
    reduce molclFieldPair force,
    reduce double * wnrg) {

if (rsq < wcut) {
    if (rsq > wlcut) {
        drsq = rsq - wlcut;
        de  = drsq * wcuti;
        de3  = de * de * de;
        dsofp = ((DESS3*de+DESS2) * de + DESS1) * de3 * TWO/drsq;
        sof = ((CESS3*de+CESS2) * de + CESS1) * de3 + ONE;
    }
    else {
        dsofp = 0;
        sof = 1;
    }
} else {
    /*nothing*/
}
Simple Analyses (3)

- In-order stream operation scheduling
  - insert stream loads and stores
  - wait until each stream operation completes
- Handle inter-cluster communication
  - SRF is sliced – each cluster can read from only a single bank of the SRF
  - need to explicitly communicate with neighboring clusters
    - simple method is to communicate through memory
    - Better way is to use the inter-cluster switch
Inter-Cluster Communication

communicate

replicate
Optimizations - multi-node (1)

- Partition streams and data
- Convert scans and reductions
- Data replication
- Task parallelism
Partition Streams and Data (1)

- Ensure load-balance
- Minimize communication
  - keep it below threshold
- Dynamically modify partition
  - data layout might change for different phases
  - schedule required movement
Partition Streams and Data(2)

**Static comm. patterns**
- Structured meshes
  - stencils
- Unstructured meshes
  - with fixed neighbor list
- Use high quality graph partitioning algorithms
  - represent communication as edges
  - computations as nodes – node weights are estimate of load
  - completely automated

**Dynamic comm. patterns**
- Hard to dynamically acquire accurate communication pattern
  - can’t use graph partitioning
- Use knowledge of the problem
  - user defined partitioning
    - streams of streams
    - define communication between the top-level stream elements (are stencils enough?)
  - geometry based
    - express geometry to the system – system has a notion of space
    - define distance metric
    - completely automated

Convert Scans and Reductions

- Optimize tree depth
- Localize if possible
  - example: neighbor-only synchronization in Jacobi
- Hide synchronization latency with work

```c
MolclSpringForces (pos0, force, &sprnrg);
Insert synchronization and tree combine
VelocUpdate (force, veloc);

kinnrg=0;
KineticEnergy(veloc,&kinnrg);
Insert synchronization and tree combine
Do on a single node:
totnrg = wnrg + sprnrg + kinnrg;

MolclSpringForces (pos0, force, &sprnrg);
kinnrg=0;
KineticEnergy(veloc,&kinnrg);
Insert synchronization and tree combines
VelocUpdate (force, veloc);

Do on a single node:
totnrg = wnrg + sprnrg + kinnrg;
```
Data Replication & TLP

- **Data replication**
  - caching
    - when to cache
    - when to invalidate
  - duplication
    - what to duplicate
    - When to move/copy memory data

- **Task parallelism**
  - current model is similar to SIMD across nodes
    - SIMD is not enforced but work is partitioned in SIMD manner
  - possibly identify task parallelism
    - might arise due to flow control – different nodes can execute different outputs of a conditional stream
Optimizations – single node

- Kernel mapping onto clusters
- Strip-mining
- Software pipelining
- Double buffering
- SRF allocation
- Stream Operation Scheduling
- Variable length streams

stream scheduling
Kernel Mapping

- **split/combine kernels**
  - optimize LRF utilization
  - kernel code and kernel dependencies

- **convert Conditionals**
  - conditional-streams
  - predication
  - higher comm. and SRF BW
  - wasted execution BW
  - tradeoff between BW and wasted execution resources

- **communication scheduling**
  - schedule FUs and buses (intra-cluster switch)
  - kernel code

- **inter-cluster communication (neighbors)**
  - utilize inter-cluster switch (comm unit)
  - reduce SRF bandwidth
  - kernel code and data layout (stencil)

- **Optimizations**
  - loop unrolling
  - software pipelining
  - kernel code

Other Optimizations (1)

Imagine
- Strip-mining
  - trade off between strip-size and amount of spilling
  - memory vs. execution BW tradeoff
    - large strips – less kernel overhead – higher exec BW
    - small strips – less spilling – higher effective mem BW

Imagine
- Software pipelining
  - hides memory latency

Imagine
- Double buffering
  - decide how much double-buffering is necessary

Imagine
- SRF allocation
  - space-time allocation
  - governs strip size and amount of spillage
  - account for and schedule memory operations

Stream lengths, kernel timing, memory bandwidth

Other Optimizations (2)

Imagine

Stream Operation Scheduling
- reorder stream operations to maximize parallelism
- insert required stream memory operations
- manage the score-board for concurrency

Imagine

Handle variable-length streams
- allocate space in local memory and SRF
- deal with timing issues