Nested Parallelism

CS149
Lecture 12

Announcements

• PA3 due today
• PA4 assigned today

A Point of View

• Parallelism is relatively easy
  - Not hard to find lots of parallelism in many apps

• The hard part is communication
  - Compute is easy
  - More difficult to ensure data is where it is needed

• Reprise: Need to make good use of resources

Sequoia

• Language: stream programming for machines with deep memory hierarchies

• Idea: Expose abstract memory hierarchy to programmer

• Implementation: benchmarks run well on many multi-level machines
  - Cell, PCs, clusters of PCs, cluster of PS3s, also + disk, GPUs

Locality

Structure algorithms as collections of independent and locality cognizant computations with well-defined working sets.

This structuring may be done at any scale.

Keep temporaries in registers
Cache/scratchpad blocking
Message passing on a cluster
Out-of-core algorithms

Efficient programs exhibit this structure at many scales.
Sequoia's Goals

- Facilitate development of locality-aware programs ... 
  ... that remain portable across machines
- Provide constructs that can be implemented efficiently
  - Place computation and data in machine
  - Explicit parallelism and communication
  - Large bulk transfers

Locality in Programming Languages

- Most languages have no notion of locality
  - Location of data is hidden by the language
- Many parallel languages provide 2 levels
  - Essentially local (private) vs. global (remote)
  - SPMD languages: UPC, Titanium
  - Array distributions: HPF, ZPL
  - Streaming languages: StreamC/KernelC, Brook
  - CUDA

Blocked Matrix Multiplication

```c
void matmul_L1(int M, int N, int T,
  float* A,
  float* B,
  float* C)
{
  for (int i=0; i<M; i++)
    for (int j=0; j<N; j++)
      for (int k=0; k<T; k++)
        C[i][j] += A[i][k] * B[k][j];
}
```

Blocked Matrix Multiplication

```c
void matmul_L2(int M, int N, int T,
  float* A,
  float* B,
  float* C)
{
  Perform series of L1 matrix multiplications.
}
```

Hierarchical Memory
Hierarchical Memory

- Abstract machines as trees of memories

Main memory

ALUs

Hierarchical Memory

- Abstract machines as trees of memories

Main memory

Aggregate cluster memory (virtual level)

L2 cache

L1 cache

ALUs

Main memory

Hierarchical Memory

Main memory

Hierarchical Memory

Main memory

Hierarchical Memory

Main memory

GPU memory

Hierarchical Memory

Aggregate cluster memory (virtual level)

Main memory

Hierarchical Memory

Main memory

Aggregate cluster memory (virtual level)
Sequoia Tasks

- Special functions called tasks are the building blocks of Sequoia programs

```c
# Sequoia Tasks

# Task args & temporaries define working set

- Task working set resident at single location in abstract machine tree

```c

task matmul::leaf( in float A[M][T],
in float B[T][N],
inout float C[M][N] )
{  
  for (int i=0; i<M; i++)
    for (int j=0; j<N; j++)
      for (int k=0; k<T; k++)
        C[i][j] += A[i][k] * B[k][j];
}
```

- Task working set resident at single location in abstract machine tree

```c
# Sequoia Tasks (Cont.)

- Sequoia parameter passing semantics are not
  - Call by value
  - Call by name

- Rather
  - Copy-in, copy-out
  - Or Call-by-value-result

- Expresses the communication of arguments and results

```c
```n
Sequoia Tasks (Cont.)

- A task says **what** is copied

- Not **how** it is copied

- The latter is machine dependent
  - File operations for a disk
  - MPI operations for a cluster
  - DMAs for Cell processor

```c

Task Hierarchies

```

```
Task Hierarchies

```c
task matmul::inner( in float A[M][T],
in float B[T][N],
inout float C[M][N] )
{
tunable int P, Q, R;
mappar( int i=0 to M/P,
      int j=0 to N/R ) {
  mapseq( int k=0 to T/Q ) {
    matmul( A[P*i:P*(i+1);P]
            B[Q*k:Q*(k+1);Q],
            C[P*i:P*(i+1);P]
            R[j:R*(j+1);R] );
  }
}
}
```

```c
task matmul::leaf( in float A[M][T],
in float B[T][N],
inout float C[M][N] )
{
  for (int i=0; i<M; i++)
    for (int j=0; j<N; j++)
      for (int k=0; k<T; k++)
        C[i][j] += A[i][k] * B[k][j];
}
```

• Tasks express multiple levels of parallelism

Task Hierarchies

Leaf Variants

• Be practical: Permit platform-specific kernels

```c
task matmul::leaf_cblas( in float A[M][T],
in float B[T][N],
inout float C[M][N] )
{
cblas_sgemm( A, M, T, B, T, N, C, M, N);
}
```

Summary: Sequoia Tasks

• Single abstraction for
  - Isolation / parallelism
  - Explicit communication / working sets
  - Expressing locality

• Sequoia programs describe hierarchies of tasks
  - Parameterized for portability

Generalizing Tasks
### Pros and Cons of Tasks

- **Isolated tasks**
  - Simplifies, improves compilation

- **Simple semantics**
  - Functional

- **If computation to communication ratio is high, can't lose**

- **Isolated tasks**
  - Task can only return
  - Cannot ask for more data, off-load results

- **Task return is a synchronization point for all sub-tasks**
  - Load balancing issue

### Discussion

- Task calls work well for *regular* problems

- Possible to compute sub-task working set before the sub-task call

  - Either
    - Sub-tasks are all about the same size
    - # of sub-tasks >> degree of machine parallelism

### Irregular Problems

- **What is an irregular problem?**

- Task working set unknown before task call
  - Need to get more input during sub-task execution
  - Can't predict size of output

- Not many sub-tasks and execution time highly variable

### In More Detail

- Must know the working set of the task at the time of the task call.

- Subtask completion is a synchronization point.

### Dynamically Determined Input

- Tasks run in isolation.

- If a child task needs additional data, it must wait for all other child tasks to finish.

### Dynamically Determined Output

- Tasks run in isolation.

- If a child task has unknown output size, it may be forced to fail or stop early.
Examples

• Dynamically determined input
  - Ray tracing
  - Or any algorithm that can benefit from caching in the memory hierarchy

• Dynamically determined output
  - Worklist algorithms
  - Or any algorithm where there may magnify its input data in data dependent ways

New Feature: Call-Up

• Tasks allow parents to invoke computation on children

• Add symmetric ability for children to invoke computation on parents

  \[
  \text{task } f(\text{in } A, \text{out } B, \text{parent } C) \{ \\
  \text{... } C.\text{foo}(X) \text{...} \\
  \}
  \]

Call-Up

Call-up runs atomically in the parent's memory.

If a child task needs additional data, it performs a call-up.

Mappar Revisited

\[
\text{task matmul::inner( in float } A[\text{M}][\text{T}], \text{ in float } B[\text{T}][\text{N}], \text{ inout float } C[\text{M}][\text{N}] ) \{ \\
  \text{tunable int } P, Q, R; \\
  \text{mappar( int } i=0 \text{ to } M/P, \text{ int } j=0 \text{ to } N/R ) \{ \\
  \text{mapseq( int } k=0 \text{ to } T/Q ) \{ \\
  \text{matmul( } A[ P*i:P*(i+1);P ][Q*k:Q*(k+1);Q], \text{ B}[Q*k:Q*(k+1);Q][R*j:R*(j+1);R], \text{ C}[P*i:P*(i+1);P][R*j:R*(j+1);R] ); \\
  \} \\
  \} \\
\}
\]

The number of iterations is known on entry to the mappar.

Spawn

• Add a parallel looping construct where the number of iterations is dynamically determined

  \[
  \text{spawn (task(\ldots), term_test);} \\
  \]

  Launch an undetermined number of task(\ldots)s. Continue until all subtasks have completed and the termination test is true.

Worklist Example

\[
\text{class Main \{} \\
  \text{Jobs workqueue;} \\
  \text{tunable childQueueSize;} \\
  \text{task worker(parent Jobs worklist)) \{ \\
  \text{Jobs myQueue;} \\
  \text{myQueue.union(worklist.subset());} \\
  \text{while(!myQueue.isEmpty()) \{ \\
  \text{Job item = myQueue.getElement();} \\
  \text{myQueue.union(item.doWork());} \\
  \} \\
  \} \\
  \}
\]

public void main() \{ \\
  // ... Some initialization work \\
  workqueue.union(initialJobs); \\
  spawn(worker(workqueue), workqueue.size()=0); \\
  // ... Some finalization work \\
  \}
\}
**Discussion**

- **Call-up**
  - Allows sub-tasks to request more input
  - Allows sub-tasks to off-load output
  - Without synchronizing with all sibling tasks

- **But**
  - Introduces explicit concurrency in parent’s address space
  - Must now reason about (atomic) interleavings of call-up in the parent

**Abstraction vs. Reality**

- The task hierarchy is abstract
- A task may have an unspecified number of sub-tasks
- The number of levels of sub-tasks may be unspecified
- Actual machines have limits in both dimensions

**Machine Descriptions**

- A separate file describes each machine
  - The number of levels of memory hierarchy
  - The amount of memory at each level
  - The number of processors at each level
- This file is written once per machine
  - Use for each program compiled for that machine

**Mappings**

- A mapping file says how a particular program is mapped on to a specific machine
  - Settings for tunables
  - Degree of parallelism for each level
  - Whether to software pipeline compute/communication

```
control(level 0)
| loop k[0]
|  splm {fullrange = 0.6; ways = 6; iterblk = 1;}
|
```

**Compilation Overview**

- The Sequoia compiler takes
  - A Sequoia program
  - A mapping file
  - A machine description
- Generates code for
  - All levels of the memory hierarchy
  - Glue to pass/return task arguments using appropriate communication primitives
Mapping Summary

- The abstract program must be made concrete for a particular machine

- Separate machine-specific parameters into:
  - Information that is common across programs
    - Machine descriptions
  - Information specific to a machine-program pair
    - Mapping files

- Mapping files can be (partially) automated