Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Task Superscalar: Using Processors as Functional Units

Yoav Etsion *Alex Ramirez* *Rosa M. Badia* *Eduard Ayguade*

Jesus Labarta *Mateo Valero*

HotPar, June 2010

Yoav Etsion
Senior Researcher

Parallel Programming is Hard

- A few key problems of parallel programming:
  1. Exposing operations that can execute in parallel
  2. Managing data synchronization
  3. Managing data transfers

- None of these exist in sequential programming…
- …but they do exists in ***processors executing sequential programs***

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

Sequential Program, Parallel Execution Engine

- Out-of-Order pipelines automatically manage a parallel substrate
  - A *heterogeneous* parallel substrate (FP, ALU, BR…)
- Yet, the input instruction stream is sequential *[Tomasulo'67][Patt'85]*

- The obvious questions:

1. ***How do out-of-order processors manage parallelism?***
2. ***Why can't ILP out-of-order pipelines scale?***
3. ***Can we apply the same principles to tasks?***
4. ***Can task pipelines scale?***

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

Outline

- **<u>Recap: How do OoO processors uncover parallelism?</u>**
- The StarSs programming model
- A high-level view of the task superscalar pipeline
- Can a task pipeline scale?
- Conclusions and Future Work

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

How Do Out-of-Order Processors Do it?

- *Exposing parallelism*
  - Register renaming tables map consumers to producers
  - Observing an instruction window to find independent instructions
- *Data synchronization*
  - Data transfers act as synchronization tokens
  - Dataflow scheduling prevents data conflicts
- *Data transfers*
  - Broadcasts tagged data
- Input is a sequential stream: complexities are hidden from programmer

Can We Scale Out-of-Order Processors?

- Building a large instruction window is difficult (**Latency related**)
  - Timing constraints require a global clock
  - Broadcast does not scale, but latency cannot tolerate switched networks
  - Broadcasting tags yields a large associative lookup in the reservation stations
- Utilizing a large instruction window
  - Control path speculation is a real problem, as most in-flight instructions are speculated (**Not latency related!**)
  - Most available parallelism used to overcome the memory wall, not exploit parallel resource (**Back to latency…**)
- But what happens if we operate on tasks rather than instructions?

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

Outline

- Recap: How do OoO processors uncover parallelism?
- **The StarSs programming model: Tasks as abstract instructions**
- High-level view of the task superscalar pipeline
- Can a task pipeline scale?
- Conclusions and Future Work

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

The StarSs Programming Model

- Tasks as the basic work unit
- Operational flow: a ***master*** thread spawns tasks, which are dispatched to multiple ***worker*** processors (aka the functional units)
- Runtime system dynamically resolves dependencies, construct the task graph, and schedules tasks
- Programmers annotate the directionality of operands
  - *input*, *output*, or *inout*
- Operands can consist of memory regions, not only scalar values
  - Further extends the pipeline capabilities
- Shameless plug: StarSs versions for SMPs and the Cell are freely available

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

The StarSs Programming Model

- Simple annotations
- All effects on shared state are explicitly expressed
- Kernels can be compiled for different processors

Intuitively Annotated Kernel Functions

```
#pragma css task input(a, b)
                 inout(c)
void sgemm_t(float a[M][M],
             float b[M][M],
             float c[M][M]);

#pragma css task inout(a)
void spotrf_t(float a[M][M]);

#pragma css task input(a) inout(b)
void strsm_t(float a[M][M],
             float b[M][M]);

#pragma css task input(a) inout(b)
void ssyrk_t(float a[M][M],
             float b[M][M]);
```

Example: Cholesky Decomposition

Barcelona Supercomputing Center Centro Nacional de Supercomputación

The StarSs Programming Model

- Code is seemingly sequential, and executes on the ***master*** thread
- Invoking kernel functions generates tasks, which are sent to the runtime
  - s2s filter injects necessary code
- Runtime dynamically constructs the task dependency graph
- Easier to debug, since execution is similar to sequential execution

Seemingly Sequential Code

```
for (int j = 0; j<N; j++) {
  for (int k = 0; k<j; k++)
    for (int i = j+1; i<N; i++)
      sgemm_t(A[i][k],
                  A[j][k], A[i][j]);

  for (int i = 0; i<j; i++)
    ssyrk_t(A[j][i], A[j][j]);

  spotrf_t(A[j][j]);

  for (int i = j+1; i<N; i++)
    strsm_t(A[j][j], A[i][j]);
}
```

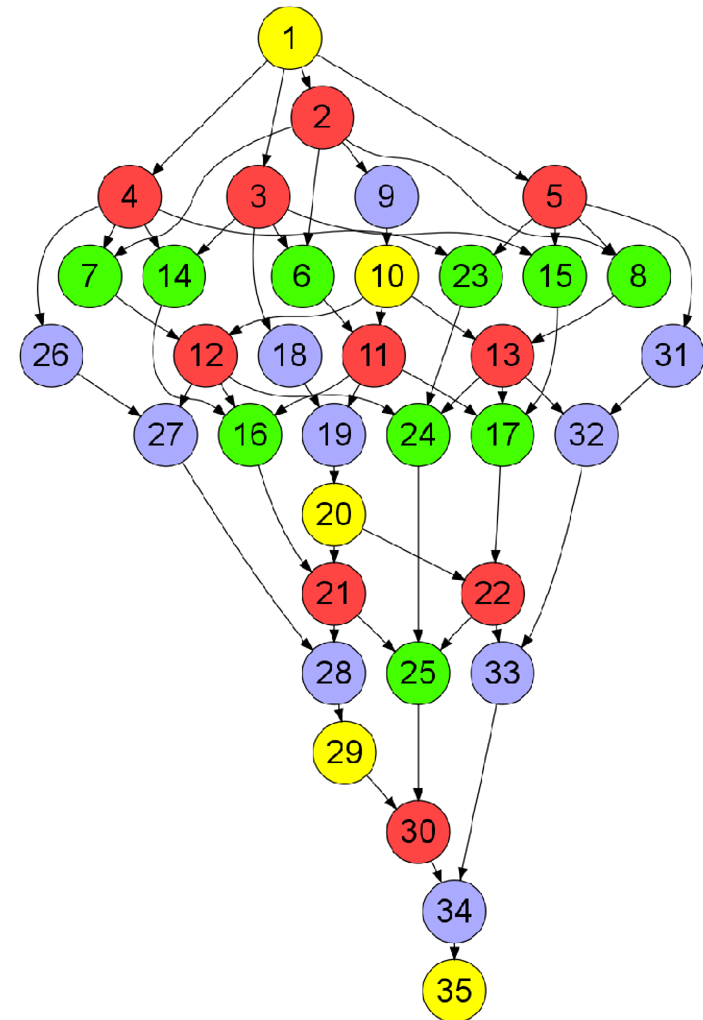
Example: Cholesky Decomposition

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

The StarSs Programming Model

- It is not feasible to have a programmer express such a graph…
- Out-of-order execution
- No loop level barriers (a-la OpenMP) Facilitates distant parallelism
- Tasks 6 and 23 execute in parallel
- Availability of data dependencies supports relaxed memory models
- DAG consistency [Blumofe'96]
- Bulk consistency [Torrellas'09]

Resulting Task Graph (5x5 matrix)



BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

So Why Move to Hardware?

- Problem: software runtime does not scale beyond 32-64 processors
- Software decode rate is 700ns - 2.5us per task
- Difference is between Intel Xeon and Cell PPU
- Scaling therefore implies much longer tasks
- Longer tasks imply larger datasets that do not fit in the cache
- Hardware offers inherent parallelism
- Vertical: pipelining
- Horizontal: distributing load over multiple units

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

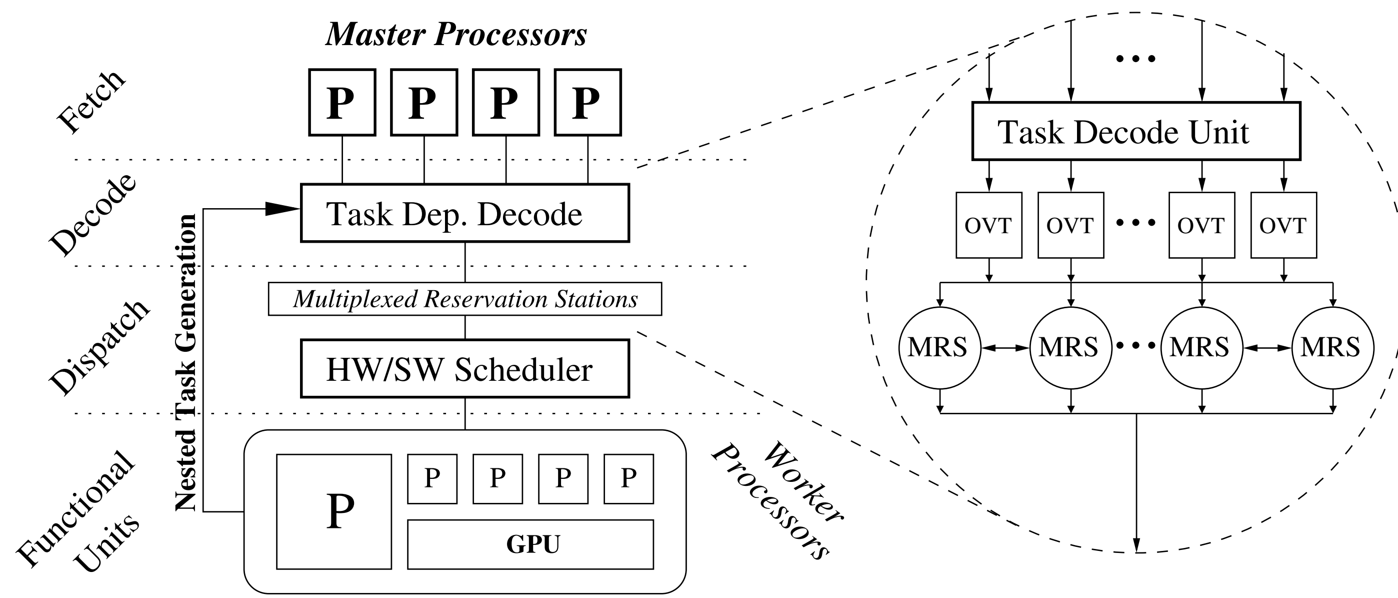
Outline

- Recap: How do OoO processors uncover parallelism?
- The StarSs programming model: Tasks as abstract instructions
- **A high-level view of the task superscalar pipeline**
- Can a task pipeline scale?
- Conclusions and Future Work

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

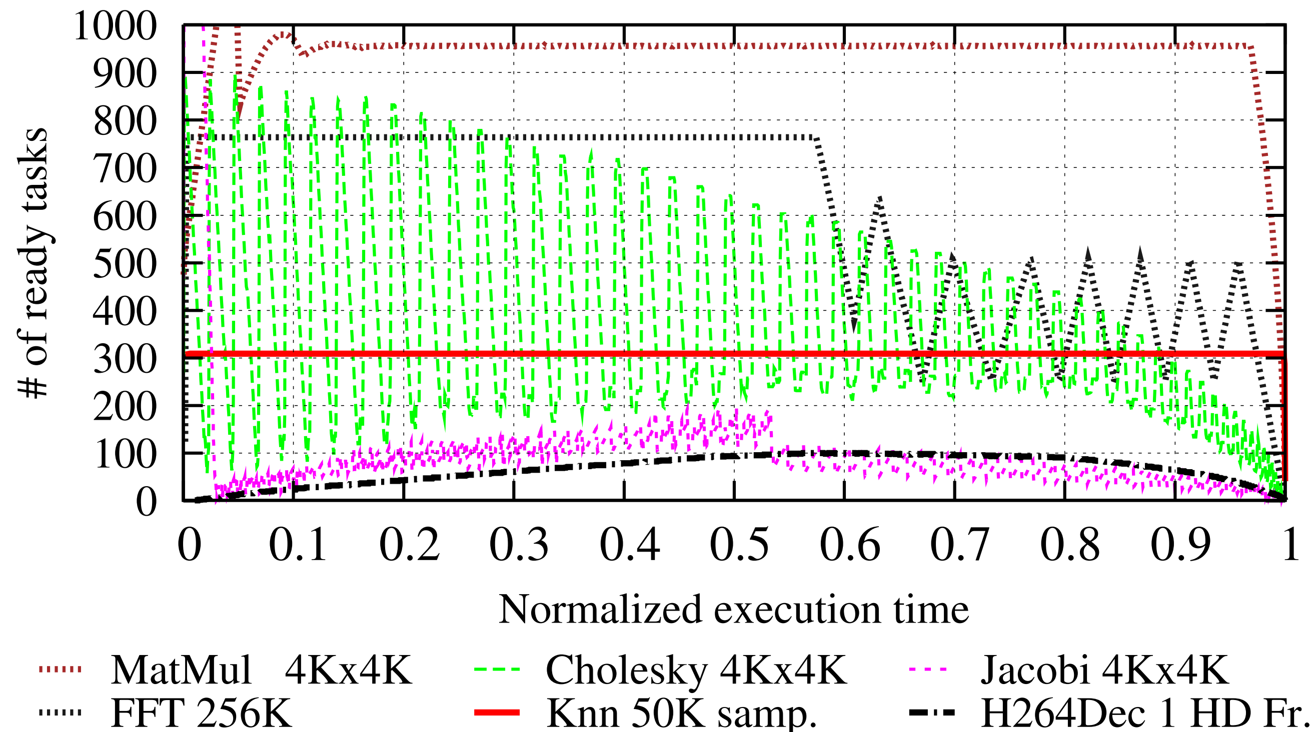
Task Superscalar: a high-level view

- Master processors send tasks to the pipeline
- Object versioning table (OVTs) are used to map data consumers and producers
  - Combination of a register file and a renaming table
- Task dependency graph is stored in multiplexed reservation stations
- Heterogeneous backend
  - GPUs become equivalent to a vector unit found in many processors



BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Result: Uncovering Large Amounts of Parallelism



- The figure shows the number of ready tasks throughout the execution
- Parallelism can be found even in complex dependency structures
  - Cholesky, H264, Jacobi

Outline

- Recap: How do OoO processors uncover parallelism?
- The StarSs programming model: Tasks as abstract instructions
- A high-level view of the task superscalar pipeline
- **Can a task pipeline scale?**
- Conclusions and Future Work

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

Overcoming the limitations of ILP pipelines

- **Task window timing**
  - No need for a global clock – we can afford crossing clock domains
- **Building a large pipeline**
  - Multiplex reservation stations into a single structure
  - Relaxed timing constraints on decodes facilitates the creation of explicit graph edges
  - Eliminates the need for associative MRS lookups
  - We estimate storing tens-of-thousands of in-flight tasks

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

Overcoming the limitations of ILP pipelines

- **Utilizing a large window**
  - Tasks are non-speculative
    - We can afford to wait for branches to be resolved
- **Overcoming the memory wall**
  - Explicit data dependency graph facilitates data scheduling
  - Overlap computation with communications
  - Schedule work to exploit locality
    - Already done on the Cell B.E. version of StarSs
  - StarSs runtime tolerates memory latencies of thousands of cycles

Outline

- Recap: How do OoO processors uncover parallelism?
- The StarSs programming model: Tasks as abstract instructions
- A high-level view of the task superscalar pipeline
- Can a task pipeline scale?
- **Conclusions and Future Work**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Conclusions

- Dynamically scheduled out-of-order pipelines are very effective in managing parallelism
  - The mechanism is effective, but limited by timing constraints
- Task-based dataflow programming models uncover runtime parallelism
  - Utilize an intuitive task abstraction
  - Intel RapidMind [McCool'06], Sequoia [Fatahalian'06], OoOJava [Jenista'10]

- Combine the two: ***Task Superscalar***
  - A task-level out-of-order pipeline using cores as functional units
  - We are currently implementing a task superscalar simulator
- Execution engine for high-level models: *Ct, CnC, MapReduce*

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

Future Work

- Explore locality-based scheduling algorithms
- HW/SW scheduling interface
- Scheduling for a heterogeneous backend
  - Automatic grouping of tasks to GPU warps
- Combining both dynamic and static dependency analysis
- Explore the effectiveness of known out-of-order optimizations
  - Lazy renaming already shown to work in StarSs

Thank You

Related Work

- Thread-Level Speculation (TLS)
  - Multiscalar [Sohi'95], Trace Procs. [Rotenberg'97], Hydra [Hammond'98]
- Dataflow
  - Intermittent work in the 1970s, 1980s, 1990s
    - [Dennis, Watson, Arvind, Culler, Papadopoulos, …]
  - TRIPS [Sankaralingam'06], WaveScalar [Swanson'03]
- Other task-level dataflow models
  - CellSs [Bellens'06], RapidMind [McCool'06], Sequoia [Fatahalian'06], OoOJava [Jenista'10]
- Hardware support for Tasks
  - Carbon [Kumar'07], ADM [Sanchez'10], MLCA [Karim'04]

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación