Enabling Technologies for a Programmable Many-core

Ben Juurlink
TU Berlin
Partner and work package leader

SEVENTH FRAMEWORK PROGRAMME

Disclaimer

- Presentation (partially) personal view on ENCORE
  - Minor focus on TU Berlin activities
- Contains some grammar mistakes
  - No time for sanity check (FP7 deadline)
  - Some grammar mistakes on purpose
    - To save space
    - ENCORE **view** matters most

Outline

- Consortium
- Objectives
- Programming Model
- Runtime System
- Preliminary Evaluation of Programming Model
- Hardware Support for Runtime System
- Conclusions & Future Work

ENCORE consortium

- Funded under FP7 Objective ICT 2009.3.6 - Computing Systems
  - 3-year STREP project (March 2010 - February 2012)

Project Objectives

- To achieve breakthrough on usability, code portability, and performance scalability of multicore systems
  - Define easy to use parallel programming model
  - Develop intelligent runtime management system
    - Hide complexity of parallel programming
      - Detect + manage parallelism
      - Detect + manage data locality
    - Hide complexity of underlying architecture
      - Heterogeneous processors
      - Physically distributed memory (NUMA)
      - Software managed memory hierarchy
  - Design scalable parallel architecture
    - Providing support to the runtime system

ENCORE Programming Model

Imperative code

```
for (i=0; i<height; i+=16)
  for (j=0; j<width; j+=16)
    mb_decode(&frame[i][j]);
```

OmpSs

```
for (i=0; i<height; i+=16)
  for (j=0; j<width; j+=16)
#pragma omp task \
    input([16][16] frame[i-16][j]) \
    input([16][16] frame[i][j-16]) \
    inout([16][16] frame[i][j])
    mb_decode(&frame[i][j]);
```

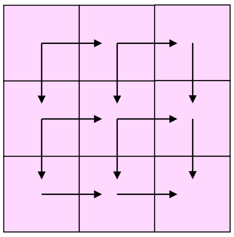
programmer

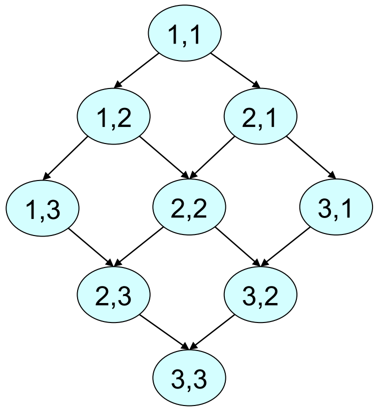
- Start from mainstream programming language (C)
- Extend sequential code with #pragma annotations
- Programmer identifies pieces of code to be executed as tasks
  - Also identifies task inputs and outputs, and specifies requirements
- Tasks need not be parallel
  - Runtime system will detect and exploit parallelism
  - Programmer is not *directly* concerned with parallelism

Task Dependency Graph

- Input/output clauses allow to build task dependency graph
  - Expressions evaluated at runtime

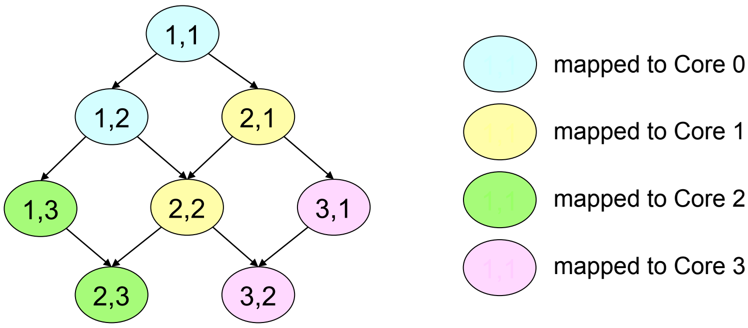
```
for (i=0; i<height; i+=16)
  for (j=0; j<width; j+=16)
#pragma omp task \
    input([16][16] frame[i-16][j]) \
    input([16][16] frame[i][j-16]) \
    inout([16][16] frame[i][j])
    mb_decode(&frame[i][j]);
```





Task Dependency Graph

- Dependency graph used by runtime system to
  - ensure correctness of execution
    - task cannot start before its predecessors have finished
  - optimize performance, e.g.,
    - reduce overhead of submitting tasks by task bundling
    - improve data locality by exploiting in/out usage information

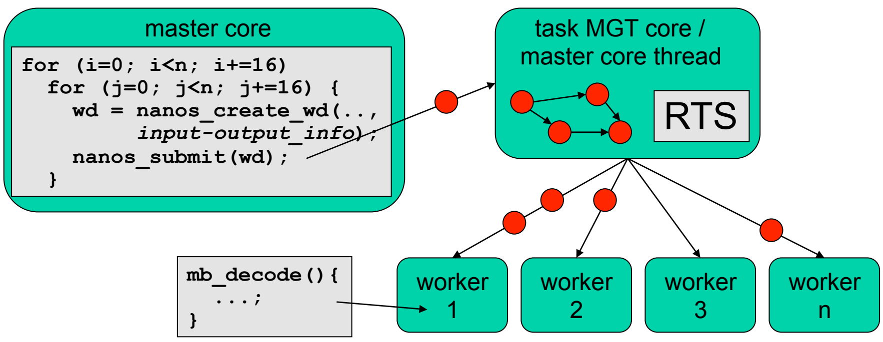


Runtime System

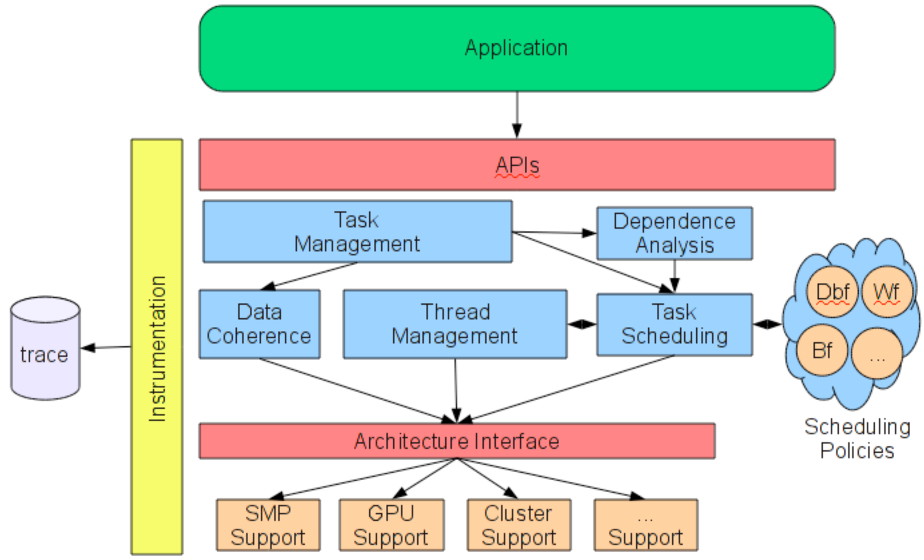
- Compiler transforms pragmas to calls to runtime system (RTS)
- Runtime system responsible for:
  - Building dependency graph
  - Extracting parallel tasks from dependency graph
  - Offloading tasks to accelerators (if applicable)
  - Managing data transfers
  - Maintaining data coherence
  - Performing optimizations while maintaining correctness
    - Task bundling
    - Memory renaming to resolve WAW and WAR hazards
    - Double buffering
    - Scheduling for locality

Execution Model

- Single master thread that submits tasks to runtime system
  - Tasks can also generate new tasks if dependency graphs disjoint
- RTS builds dependency graph and submits tasks to worker cores
- Worker cores execute tasks and request RTS new tasks when done



Runtime Library Structure



Supported Platforms

- SMP
- SMP-NUMA
  - Makes copies of input/output data in local memory
- SMP-Cluster
  - Makes copies across the network
- CUDA
  - Manages copies to/from GPUs with overlapping
- ENCORE

Preliminary Performance Evaluation

- How well does OmpSs perform on non-HPC applications?

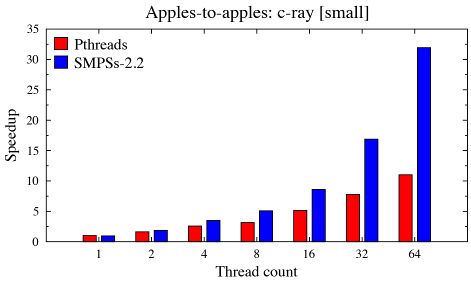
- Next performance evaluation uses SMPSs
  - SMP-instance of StarSs
  - StarSs subset of OmpSs features
- Performance evaluation preliminary
  - SMPSs startup cost not included (=large, negligible for large applications)
  - Still need to analyze results in detail
- “Non-biased” comparison
  - TU Berlin not involved in SMPSs development

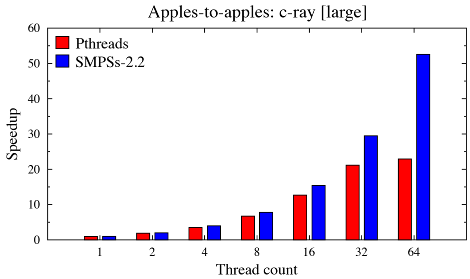
Experimental Setup

- Platform:
  - 64-core cc-NUMA
  - HP DL980 G7
    - 8x Xeon X7560 (Nehalem EX)
- Benchmarks:
  - Kernels: mainly from EEMBC MultiBench
  - Applications: H.264 decoding
  - Workloads: set of several kernels/applications
- Methodology:
  - Started with EEMBC MultiBench
  - Stripped away MITH framework
  - Ported to Pthreads
  - Ported to SMPSs
- Compare SMPSs to Pthreads

C-ray Kernel

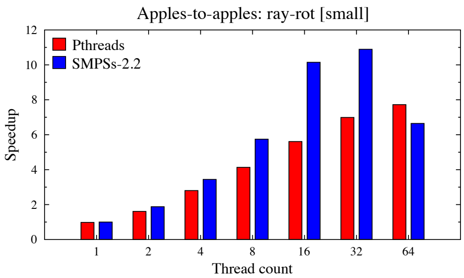
- Brute force raytracer
- 500 (SMPSs) / 700 (Pthreads) LoC
- Unoptimized, simple, clean
- Distributes (blocks of) scanlines to workers

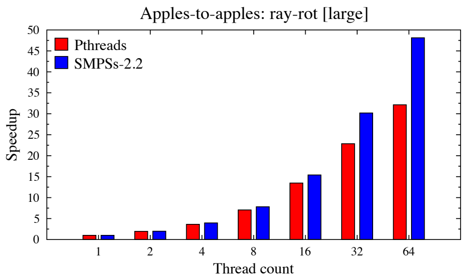




Ray-Rot Workload

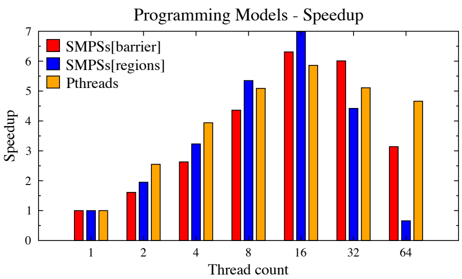
- C-ray feeds binary output to rotate kernel
- Pipelining parallelism (easier to exploit in SMPSs)
- Introduces additional dependencies
- Rotation angle is 90°

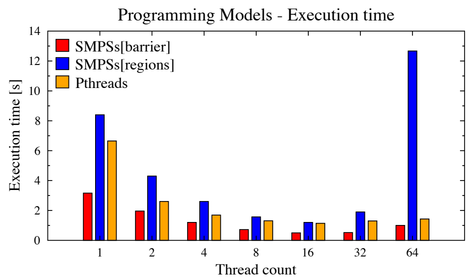




Rot-cc Workload

- Rotate feeds binary output to rgbcmy kernel
- Pipelined, dependent, requires regions
- Cache performance deteriorates
- Rotation angle is 90°



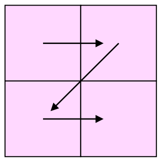


Preliminary Conclusions from Preliminary Performance Evaluation

- OmpSs / SMPSs is good
  - For several benchmarks SMPSs performs better than Pthreads
  - Serial program behavior maintained
  - (Often) programs just ‘work’ after adding pragmas
  - Very easy to exploit DLP using task-level parallelism
- Task-based parallel programming model in development
  - Documentation can be improved
  - Compiler does not support all constructs
  - Parameter list ‘explosion’
  - Programming style restrictions (syntax / structure) (bad?)

Architecture Support for Runtime System

- In OmpSs / StarSs, runtime takes care of
  - Task dependency determination
    - Task B depends on task A if output of A overlaps input of B
  - Scheduling while
    - Reducing task issuing overhead
    - Optimizing data locality
- This can take a lot of time
  - Reduces scalability when threads are fine grain
  - Coarse grain threads reduce scalability also
    - Lose-lose situation
- Next evaluation performed using CellSs
  - Cell instance of StarSs
- "Complex dependencies (CD)" pattern
  - H.264-like dependencies



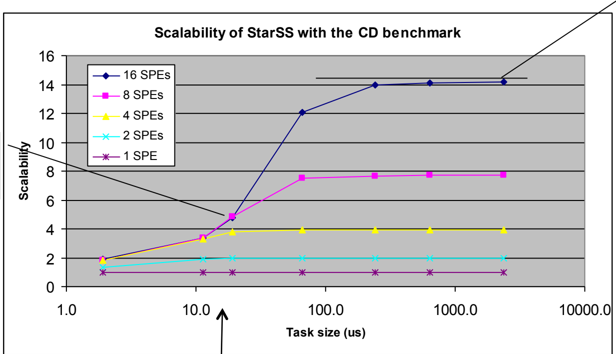
encore

Scalability of CellSs Runtime System

- "Optimal" CellSs configuration

max = 14.5

Scalability = 4.9 max

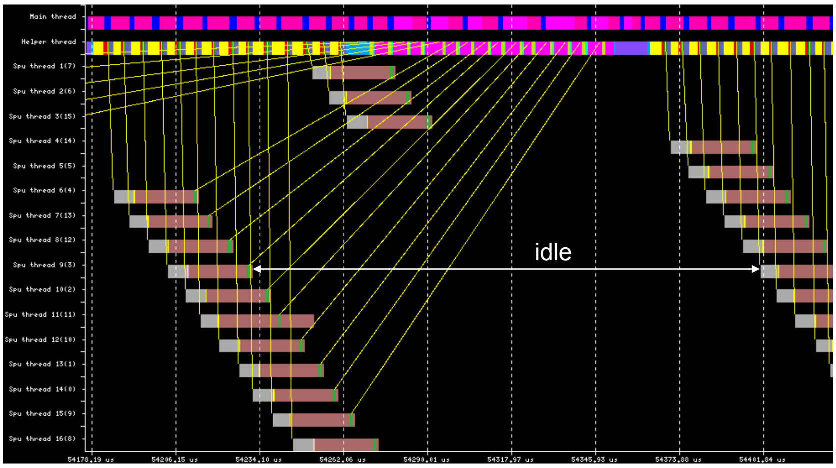


H.264 MB decoding:
Average = 20μs

encore

Scalability of CellSs

Paraver trace of CD (task size 19µs)



encore

Nexus: HW Support for TPU

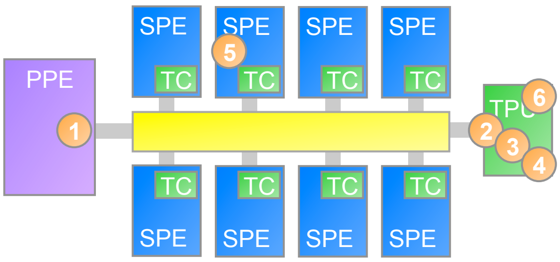
Task "life cycle":

1. Create task descriptor and send its address to TPU.
2. Load task descriptor.
3. Process task descriptor; update task pool
4. Add ready tasks to ready queue.
5. Read ready queue; process; inform TPU.
6. Update task pool.

Task Descriptor

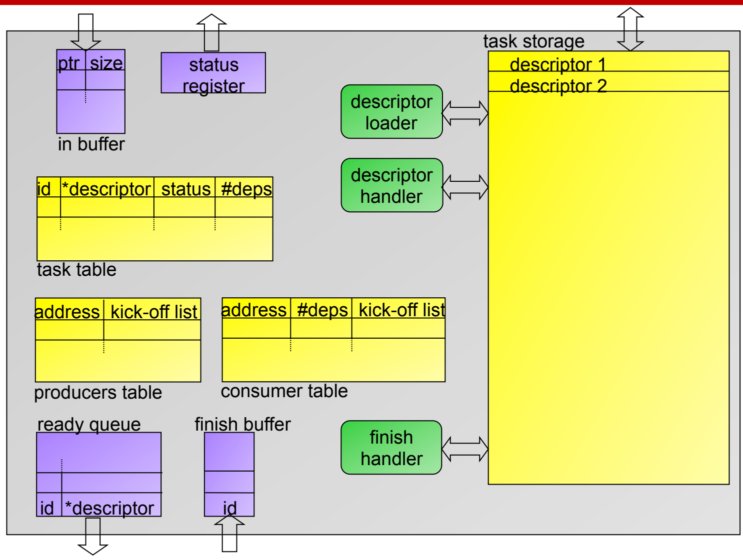
task_func
no_params
p1_io_type
p1_pointer
p1_x_length
p1_y_lenght
p1_y_stride
p2_io_type
…

Pipelined for throughput

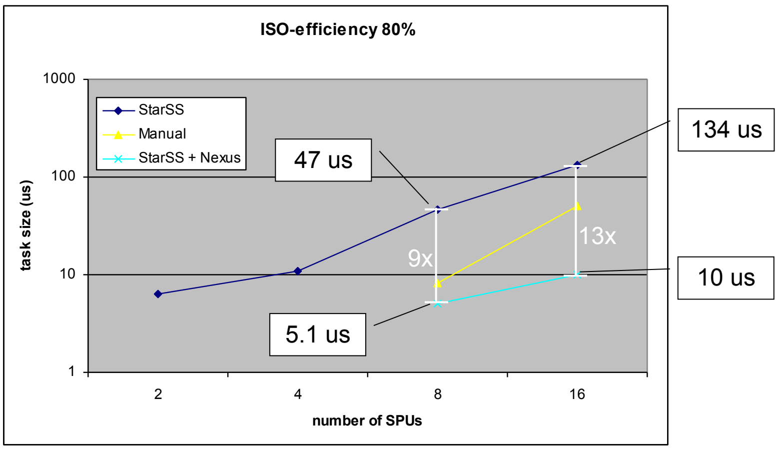


encore

Nexus TPU Design



Preliminary Evaluation Results for Nexus



Preliminary Conclusions on Nexus

- Runtime System of CellSs / OmpSs can become bottleneck
  - Mainly for fine-grain tasks
- HW support (Nexus) can remove bottleneck
  - Up to 100+ (?) cores
- Detailed VHDL model will be designed, implemented, and evaluated in ENCORE

Conclusions

- ENCORE targets
  - Programmability
  - Performance portability
  - Right kind of hardware support
- Preliminary SMPSs vs. Pthreads comparison shows
  - Satisfactory performance achieved with little programming effort
- Preliminary Nexus task manager
  - Runtime system not bottleneck until 100+ cores

Future Work in ENCORE

- Programming model
  - Region dependency checking
    - Allows to capture more complex dependency patterns
  - Improve runtime scheduling
    - Based on locality
    - Based on QoS
- Applications and performance evaluation
  - Can we effectively and efficiently implement H.264 decoding in OMPSs?
- Hardware support for runtime system
  - VHDL model of Nexus++ in FPGA multicore prototype
- . . .

- Stay tuned at http://www.encore-project.eu

Backup Slides

Heterogeneity

```
#pragma omp task input([BS][BS] A, [BS][BS] B) inout([BS][BS] C)
void matmul(float *A, float *B, float *C) {
   // original sequential matmul
}

#pragma omp target device(cuda) implements(matmul) copy_deps
void matmul_cuda (float *A, float *B, float *C) {
   // optimized kernel for cuda
}

// library function
#pragma omp target device(cell) implements(matmul) copy_deps
void matmul_spe(float *A, float *B, float *C);
```