Dynamic Multigrain Parallelization on the Cell Broadband Engine

Filip Blagojevic, Dimitrios S. Nikolopoulos, Alexandros Stamatakis and Christos D. Antonopoulos

Nathan Tallent

(Slides adapted from the authors.)
Introduction and Motivation

- **Cell BE:**
  - 1 general purpose proc. (PPE)
  - 8 special vector proc. (SPE)
  - Element Intercon. Bus (EIB)

- **LS** – software managed using DMA req.

- **Challenges** with scientific apps on Cell:
  - Cell designed for single precision FP
  - SPEs lack branch predictor
  - Heterogeneous architecture with multiple levels of parallelism
Introduction and Motivation

- Different levels of parallelism
  - PPE is 2-way SMT
  - 8 SPEs – software managed cache
  - SPE – DMA – computation overlap
  - SPE – DLP, vector registers
  - SPE – ILP, double pipeline

- Peak performance must use *all* SPEs
  - How to map an application to SPEs?
  - What to execute on the PPE?

Cell is powerful on paper… but hard to harness.
Overview

RAxML
- bioinformatics application
- Porting RAxML to Cell

Static parallelization schemes
- TLP – Task Level Parallelism
- LLP – Loop Level Parallelism

Dynamic (adaptive) parallelization scheme
- combines TLP and LLP
RAxML: All you need to know

- Bioinformatics application

- Computational characteristics
  - Computationally **intensive**
  - *Embarrassingly parallel*

- MPI version – **master-worker** architecture
  - Each worker independently performs *bootstrapping*
  - *bootstrap* = independent task
  - Minimum 100 bootstraps required
Porting RAxML to Cell

- Identify functions to off-load to SPE

- SPE-specific
  - Replace math lib calls with Cell-optimized versions
  - Vectorize expensive conditional statements
  - Double buffer DMA requests: overlap communication and computation
  - Vectorize computations

- Optimize PPE-SPE communication

Performance improvement of 2.65 over naively off-loaded SPE code
Overview

- RAxML
  - bioinformatics application
  - Porting RAxML to Cell

→ Static parallelization schemes
  - TLP – Task Level Parallelism
  - LLP – Loop Level Parallelism

- Dynamic (adaptive) parallelization scheme
  - combines TLP and LLP
Mapping Processes to Cell: Intro

- Initial mapping: one MPI process to each PPE thread
Initial mapping: one MPI process to each PPE thread
• Consequence: Only 2 SPEs used

(Linux scheduler)
Task-Level Parallelization (TLP)

- Up to 8 MPI processes executed on the PPE
• Off-loaded functions parallelized across SPEs
• Off-loaded functions parallelized across SPEs

Loop work-sharing, like OpenMP
Multigrain Parallelization

- Off-loaded functions parallelized across SPEs

Loop work-sharing, like OpenMP
Task-Level parallelization (TLP)

- Terminology
  - **Task** – Off-loadable function from an MPI process
  - **Event** – Off-loading a single task

- Tasks served by a user-level event-driven scheduler
  - Scheduler waits for events (task off-loads)
  - Fair task generation across processes (round-robin)
  - Scheduler prevents the PPE from blocking
    - (Linux effectively blocks -- too large a quantum)

- Scheduler oversubscribes PPE to increase available SPE parallelism
Event-Driven scheduler (ED TLP)

- Simplified model: 1 PPE

<table>
<thead>
<tr>
<th></th>
<th>EDTLP</th>
<th>LINUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SPE</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

key: offload and switch (much smaller quantum)
Scheduling Loop-Level Parallelism (LLP)

- Off-loaded tasks are split across SPEs
  - Independent loop iterations are distributed across SPEs (cf. OpenMP)
  - Number of SPEs is user or system controlled
LLP - Performance

- Scaling across SPEs
  - 1 process, 1 bootstrap
  - ~250 loop iterations

- Using 5+ SPEs decreases efficiency:
  - Fine granularity of the loops
  - Load imbalance due to memory alignment constraints
Multi-Grain Parallelism I

- TLP + LLP

No programming model performs the best in all situations
Multi-Grain Parallelism II

![Graph showing execution time in seconds vs. number of bootstraps for EDTLP+LLP with 4 SPEs and 2 SPEs per parallel loop, as well as EDTLP.]
Overview

- RAxML
  - bioinformatics application
  - Porting RAxML to Cell

- Static parallelization schemes
  - TLP – Task Level Parallelism
  - LLP – Loop Level Parallelism

- Dynamic (adaptive) parallelization scheme
  - combines TLP and LLP
Multi-Grain Parallel Scheduler (MGPS)

- MGPS extends the Event-Driven scheduler
  - adaptive scheduling policy
  - distributed: attached to each process

- Scheduler is invoked on
  - Task off-loading request (arrival)
  - Off-loaded task completion (departure)

- Upon departure, possible change of parallelization strategy
• Starts with 8 MPI processes
MGPS II

- Application only has enough work for 2 processes
The MGPS scheduler activates LLP model
MGPS Implementation

- Maintain **dual** copies of off-loaded functions
  - One for Task-Level Parallelism
  - One for Loop-Level Parallelism
  - (Along with PPE version)

- Disadvantage:
  - increased code size

- Advantage:
  - fewer conditionals (very expensive on SPEs)
Scheduling Policies–Comparison I

![Graph showing execution time in seconds vs. number of bootstraps for different scheduling policies.](image)
Scheduling Policies–Comparison II

![Graph showing the comparison of execution times for different scheduling policies. The x-axis represents the number of bootstraps, and the y-axis represents execution time in seconds. The graph compares MGPS, EDTLP+LLP with 4 SPEs per parallel loop, EDTLP+LLP with 2 SPEs per parallel loop, and EDTLP.](image)
Parallelization Across Multiple Cells I
Parallelization Across Multiple Cells II

- MGPS
- EDTLP+LLP with 4 SPEs per parallel loop
- EDTLP+LLP with 2 SPEs per parallel loop
- EDTLP

Graph showing execution time in seconds vs. number of bootstraps.
Who cares about MGPS?

- Claim: Since # tasks is large, simple EDTLP is fine
  - MGPS converges to EDTLP when # tasks is large
  - LLP requires (relatively) high overheads
  - This is the ‘normal’ case

- However, consider a large cluster of Cells (peta-scale)
  - Inter-node communication is expensive
  - Probably better to tie a few tasks to one Cell
  - MGPS (adaptation) may be useful
Conclusions/Contributions

- Different strategies required to schedule multi-level parallelism on the Cell BE

- Explored profitability of loop-level parallelism

- Proposed a dynamic scheduling policy for combining task-level and loop-level parallelism

- On RAxML, our scheduler outperforms native OS by 2.6x
  - Linux (task level)

- [Well written (no tortured grammar/phrases!)]

Comparison With Other Processors I

- Intel Pentium 4 Xeon
  - 2-way SMT
  - Running at 2GHz
  - 8KB L1-D cache, 512KBL2 cache, and 1MB L3 cache
  - 2 processors used

- Power5
  - Dual core
  - Each core 2-way SMT
  - Running at 1.6 GHz
  - 32KB of L1-D and L1-I cache, 1.92 MB of L2 cache, and 36 MB of L3 cache
Comparison With Other Processors II

- Cell is faster
  - Outperforms the Intel Xeon by a large margin
  - Performs slightly (5–10%) better than the IBM Power5
All Done