MapReduce on the Cell Broadband Engine Architecture

Marc de Kruijf
Overview

- Motivation
  - MapReduce
  - Cell BE Architecture
- Design
- Performance Analysis
- Implementation Status
- Future Work
What is MapReduce?

- A parallel programming model for large-scale data processing
  - Simple, abstract interface
  - Runtime handles all synchronization, communication, and scheduling.
- Implementations exist for:
  - Large distributed clusters, as in Google’s original MapReduce
  - Shared memory multiprocessors, as in Stanford’s Pheonix
- The Cell processor is neither of these….
What is the Cell then?

The Cell is:

“A single chip multiprocessor with nine processors operating on a shared, coherent memory”

So what’s the difference?

From a programming perspective, the Cell is much more like a “cluster-on-a-chip”

That sounds hard…

It’s not easy.
Motivation

- Programming the Cell is hard… yet…
  - Distributed (shared?) memory single-chip multiprocessors are the way of the future. (Opinion.)
  - Corollary: Shared memory is out. Message passing is in. (More opinion.)

- What’s missing are the right runtime and abstraction layers to enable the scalability potential of these and future systems.
  - MapReduce is just such an abstraction.
Overview

- Motivation
  - MapReduce
  - Cell BE Architecture
- Design
- Performance Analysis
- Implementation Status
- Future Work
MapReduce Refresher

Map

to,1 | a,1 | man,1
dog,1 the,1 | to,1
dog,1 | to,1
to,1 | the,1

Group by Key

Reduce

a,1 dog,1 man,1 the,1:1 to,1:1:1

a,1 dog,1 man,1 the,2 to,3
MapReduce on Cell

- **Design**
- **Execution variants:**
  - MapReduce, sorted
    - Phases 1-5
  - MapReduce, no sort
    - Phases 1-4
  - Map only, sorted
    - Phases 1, 2, and 5
  - Map only, no sort
    - Only phase 1
MapReduce on Cell

- **Design**
- **Execution variants:**
  - MapReduce, sorted
    - Phases 1-5
  - MapReduce, no sort
    - Phases 1-4
  - Map only, sorted
    - Phases 1, 2, and 5
  - Map only, no sort
    - Only phase 1
MapReduce on Cell

- **Design**
- **Execution variants:**
  - MapReduce, sorted
    - Phases 1-5
  - MapReduce, no sort
    - Phases 1-4
  - Map only, sorted
    - Phases 1, 2, and 5
  - Map only, no sort
    - Phase 1 (no hash)
MapReduce on Cell

- **Design**
- **Execution variants:**
  - MapReduce, sorted
    - Phases 1-5
  - MapReduce, no sort
    - Phases 1-4
  - Map only, sorted
    - Phases 1, 2, and 5
  - Map only, no sort
    - Phase 1 (no hash)
**MapReduce on Cell**

- **Design**
- **Execution variants:**
  - MapReduce, sorted
    - Phases 1-5
  - MapReduce, no sort
    - Phases 1-4
  - Map only, sorted
    - Phases 1, 2, and 5
  - Map only, no sort
    - Phase 1 (no hash)
MapReduce on Cell

- **Design**

- **Execution variants:**
  - MapReduce, sorted
    - Phases 1-5
  - MapReduce, no sort
    - Phases 1-4
  - Map only, sorted
    - Phases 1, 2, and 5
  - Map only, no sort
    - Phase 1 (no hash)
Design Highlights

- Output is pre-allocated.
  - Enhances performance as well as output locality.

- Work queue allows dynamic scheduling of tasks among SPEs for load balancing.
  - Adding priorities allows pipelining of computation to maximize resource utilization.

- Outside of DMA transfers, there is no data copying.
Overview

- Motivation
  - MapReduce
  - Cell BE Architecture
- Design
- Performance Analysis
- Implementation Status
- Future Work
Performance Analysis

- Assumptions:
  - While there is scheduled work for the SPEs, the SPEs are always the bottleneck.

- Execution time:
  - Fixed runtime startup cost = unknown (can be amortized) +
  - Map execution time = (# map keys * map execution time) +
  - Sort time = $\frac{n \log(n)}{2}$, where $n = (#$ intermediate keys / hash range) +
  - Reduce execution time = (# reduce keys * reduce execution time) +
  - Sort time = $\frac{n \log(n)}{2}$, where $n = (#$ reduce keys / output buffer size)

- Observations:
  - Buffer management and partitioning is a necessary part of programming the SPEs and is not considered overhead
  - Sorting is the dominant overhead… let’s examine it further.
Performance Analysis

- **Sorting**
  - Q: I hear the SPEs are not good at control tasks, but sorting is a control task, isn’t it?
  - A: You are right, let’s analyze the sorting performance of SPEs vs. a typical superscalar.

- **Assumptions**
  - Let’s assume our key comparison function is string compare.
  - Furthermore assume that our input strings are uniformly distributed.

- **SPE `strcmp()` average exec time / comparison**: 36 cycles
  - Assuming 3.2 Ghz: 11.25ns / comp

- **x86 `strcmp()` average inst / comparison**: 28 inst
  - Assuming IPC of 1.5 @ 3.2 Ghz: 5.8ns / comp

- **Bottom line**: we have 8 SPEs to 2 (maybe 4) x86 cores:
  - 11.25 / 8 = **1.406 ns/comp (CELL)**
  - 5.8 / 4 = **1.45 ns/comp (Kentsfield)**

```c
/* strcmp */
int keyCompare(const void *one, const void *two)
{
    char *a1 = (char *)one;
    char *a2 = (char *)two;
    int i;

    for (i = 0;
        _builtin_expect(a1[i] == a2[i], 0) &&
        _builtin_expect(a1[i] != 0), 1);

    if (__builtin_expect(a1[i] == a2[i], 0)
        return 0;
    else if (a1[i] > a2[i])
        return 1;
    else
        return -1;
}
```
Overview

- Motivation
  - MapReduce
  - Cell BE Architecture
- Design
- Performance Analysis
- Implementation Status
- Future Work
Implementation Status

- Very simple test application runs to completion
  - Larger test program does not return. Debugging using the simulator is erratic and frequently ends with a timeout.
  - Overestimated the capabilities of the simulator.
  - Cell Blade time needed, but too late to bear fruit.

- ~2500 loc vs. ~1200 for Phoenix (shared mem. implementation)
  - 21 threads total, though synchronization is not too difficult
  - Buffer management is the hard part…
Future Work

- Wrap up implementation
  - Add performance counters to quantify overhead.
- Perform application-agnostic runtime analysis
  - Combine with static analysis to determine performance bottlenecks
  - Determine if hash should be on the PPE or SPE
- Perform application-specific runtime analysis
  - What happened to Marching Cubes??
    - Actually mostly done, but no opportunity to test