BlockLib: A Skeleton Library for Cell Broadband Engine™

Markus Ålind, Mattias V. Eriksson, Christoph Kessler

PELAB
Linköping university
Sweden


CELL BE Processor (IBM, Sony, Toshiba)

PPE: Master processor
- Power Processor Element, 64bit dual-thread PowerPC, with L1 and L2 cache
- Good at control tasks, task switching, OS-level code

8 slave processors (SPE)
- Synergistic Processor Element
- Good at compute-intensive tasks
  - 1 mult+add on 4 floats per cc (SIMD)
  - 1 mult+add on 2 doubles per 7 cc
- Total, 204 GFLOPs peak
- Small local memory 256KB (code and data)
- No direct access to main memory – need DMA transfers
- EIB total BW 96 bytes/cc

Issues in writing efficient SPE code
- C, IBM Cell SDK
- Local store
  - 256 kB
- Explicit DMA transfers
  - Between main memory and local store
  - Between SPEs
  - Asynchronous
  - 16 byte aligned to work at all, 128 byte aligned to be fast
- Multi buffering
- SPE Calculations
  - 16 byte internally
  - Loads and stores are always 16 byte
  - Vector of 4 floats faster than single scalar
Skeleton programming

- **Skeletons**
  - For common computation patterns
  - For which efficient parallel implementations may exist
    - Data-parallel skeletons: e.g. map, reduce, scan, map-with-overlap
    - Task-parallel skeletons: e.g. farm, pipe, while
  - Generic
  - Compositional

- **Goals**
  - Abstraction
  - Hide implementation
  - Hide parallelization
  - "Parallel programming as easy as sequential programming"
  - Code reuse

BSP model

- **Bulk-synchronous parallelism**
- **Supersteps**

NestStep

- Global address space language for the CRCW BSP model
- Locality-aware shared memory abstraction for distributed memory systems
  - Shared variables and arrays
  - Distributed shared arrays (a la HPF, UPC, …)
  - Private variables and arrays
- Programmable combining at the end of each superstep
- Runtime system
  - manages threads, memory, communication
  - Implemented for MPI clusters and CELL BE

BlockLib

- Set of data-parallel skeletons
  - Map
    - `map op (x_1,...,x_n) := ( op (x_1), ..., op (x_n) )`
    - Also with multiple argument vectors
  - Reduce
    - `reduce op (x_1,...,x_n) := x_1 op x_2 op ... op x_n`
  - Map-Reduce
  - Map-with-overlap

- Each skeleton instance (call) can be a stand-alone superstep
- or called within a superstep by a SPE thread group

Realization 1: Generic functions

- Code parameter op passed via function pointer
  - Very high overhead on SPE
- Improvement 1a: User provided inner loop
- Improvement 1b: User hand SIMD optimizing
- Minor code bloat
- Array Elements Summation: 4 lines in normal C code, 21 lines SIMD-optimized,
  - Function definition - automatic optimization

Realization 2: Generative approach

- Define code operand op as sequence of predefined macros
  - one per elementary arithmetic operator in op
  - and base type (float, double)
  - easily extensible
  - expand to SPE special instructions where possible
- Code generation macros for integrated skeleton function generation
- Macro expansion by C preprocessor

```c
// Standard C
int i;
float sum = 0;
for (i=0;i<N;i++)
sum += x[i];

// Definition
DEF_REDUCE_FUNC(SUM(my_sum, t1, BL_NONE, BLSADD(t1, op1, op2))
// C expression
res = my_sum(s, N);
```
**BlockLib Implementation Details**
- Inter-SPE communication with DMA
  - Special signal register
  - Message passing
- Synchronisation
  - Group synchronisation
  - Data combine

**Evaluation**
- Synthetic benchmarks
- Real application: Parallel ODE solver
  - Usability
  - Absolute performance
- Playstation 3
  - 6 SPEs
  - 210 MiB RAM
  - Vector size 5M elements each

**Micro-Benchmark**
- Dot product (float)
  - 5.3 GFLOPS = 21.2 GB/s

**Micro-Benchmark**
- Compute-intensive code parameter
- 24 ops per 3 elements
- 1 op per element

**Comparison to IBM Cell BLAS-1 Library**
- Dot product

**ODE Solver application**
- Vector based scientific program
- Based on LibSolve
  by Matthias Korch and Thomas Rauber (Univ. Bayreuth)
- Brusselator equations
  - Spatial discretization of partial differential equations by the method of lines
  - Limited access distance - fits map-with-overlap
  - The rest is simple loops and reductions - fits map, reduce or map-reduce
- Double precision
ODE code excerpt, using BlockLib

```c
// original code
error_max = 0.0;
for (i = 0; i < ode.size; ++i)
    temp = fabs(err_vector[i] / yscal[i]);
if (temp > error_max)
    error_max = temp;
// BlockLib definition
ODE_MAP_TWO_AND_REDUCE_FUNC(D(dDistMaxAbsQuot,
    abs,
    max,
    BL_NONE,
    BLDIV(div, m_op1, m_op2)
    BL_ABS(abs, div),
    BLDMA(X(max, r_op1, r_op2));
// usage
error_max = dDistMaxAbsQuot(err_vector, yscal, ode.size)
```

ODE Solver Execution Times

![Graph of ODE Solver Execution Times](image)

ODE Solver Speedup

![Graph of ODE Solver Speedup](image)

ODE Solver, Sources of Overhead

Figure 10: Time distribution of ODE solver with ode size 200000.

Figure 11: Time distribution of ODE solver with ode size 125000.

ODE Solver, Comparison

Execution times for the minimal sequential solver on x86 (optimized) and the Cell port (using six SPEs)

<table>
<thead>
<tr>
<th>machine</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4 (K,R)</td>
<td>2196</td>
</tr>
<tr>
<td>Opteron (K,R)</td>
<td>1979</td>
</tr>
<tr>
<td>P4 Xeon, GCC</td>
<td>1626.17</td>
</tr>
<tr>
<td>P4 Xeon, ICC</td>
<td>1520.23</td>
</tr>
<tr>
<td>Core 2, GCC</td>
<td>1386.56</td>
</tr>
<tr>
<td>Cell, 6 SPE:s, GCC</td>
<td>303.14</td>
</tr>
</tbody>
</table>

ODE Solver, Synchronization Cost

Speedup using NestSteps native superstep synchronization

Speedup using BlockLib synchronization
BlockLib Summary

- Skeleton programming library for Cell
  - Abstraction from Cell SPE programming complexity
- Called from NestStep programs or used stand-alone
- Data-parallel skeletons implemented:
  - map, reduce, map+reduce, map-with-overlap
- Variants
  - Generic functions
    - optionally with user-supplied optimized kernels
  - Generative approach
- Outperforms IBM Cell SDK 3.0 BLAS-1 dot product for p>2
- Good speedup on ODE solver application

References