Implementation of NAMD molecular dynamics non-bonded force-field on the Cell Broadband Engine processor

Guochun Shi (gshi@ncsa.uiuc.edu), *Volodymyr Kindratenko* (kindr@ncsa.uiuc.edu)
National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign

Cell Processor and IBM Cell Blade

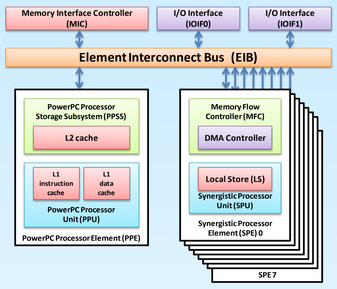


Figure 1: Cell/B.E. processor architecture

- One Power Processor Element (PPE) and eight Synergistic Processing Elements (SPEs)
- SPE's local storage (LS): 256 KB Processor clock speed: 3.2 GHz
- 25.6 GB/s processor-to-memory bandwidth
- 205 GB/s EIB sustained aggregate bandwidth
- Theoretical peak performance: 204.8 GFLOPS (SP) and 14.63 GFLOPS (DP)

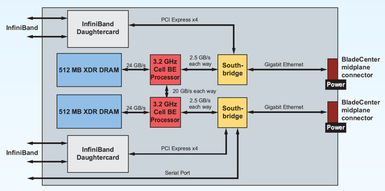


Figure 2: IBM dual-Cell/B.E. blade

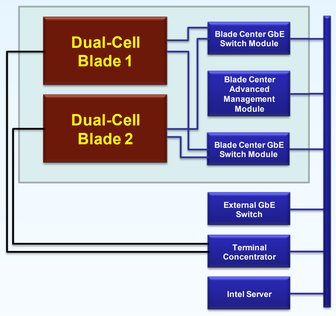


Figure 3: IBM Cell/B.E. blade center

- 410/28 GFLOPS (SP/DP) peak
- 512 MB of XDR DRAM per Cell/B.E. processor
- Gigabit Ethernet

Function offload programming model

- User compute task structure inherits the common *task_t* structure:

```
typedef struct task_s {
      int cmd; // operand
      int size; // size of task structure
} task_t;
```

```
typedef struct compute_task_s {
      task_t common;
       <user_type1> <user_var_name1>
       <user_type2> <user_var_name2>
        …
} compute_task_t;
```

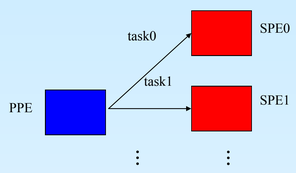


Figure 4: Task based function offload model

- API for the PPE

```
int ppu_task_init(int argc, char **argv,spe_program_handle_t );  // initialization
int ppu_task_run(volatile task_t * task);  // start a task in all SPEs
int ppu_task_spu_run(volatile task_t* task, int spe); // start a task in one SPE
int ppu_task_spu_wait(void); // wait for any SPE to finish, blocking call
void ppu_task_spu_waitall(void); // wait for all SPEs to finish, blocking call
```

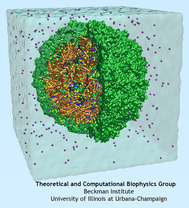
- API for the SPEs

```
int spu_task_init(unsigned long long);
int spu_task_register(dotask_t,int);  // register a task
int spu_task_run(void);  // start the infinite loop, wait for tasks
```

- Two programming models are supported: *single SPE* and *multiple SPEs*

NAMD kernel

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems.



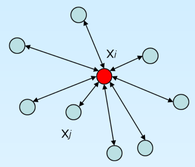
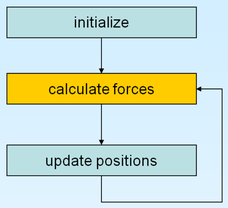


Figure 5: MD simulation principles



x_i^k — time step; atom index

$$F(x_i^k) := \sum_{i \neq j=1}^{N} f(x_i^k, x_j^k)$$

$$x_i^{k+1} = x_i^k + f(F(x_i^k))$$

NAMD SPEC 2006 CPU benchmark kernel:

```
1: for each atom i in patch p_k
2:       for each atom j in patch p_l
3:             if atoms i and j are bonded, compute bonded forces
4:             otherwise, if atoms i and j are within the cutoff distance, add atom j to the i's atom pair list
5:       end
6:       for each atom k in the i's atom pair list
7:             compute non-bonded forces (L-J potential and PME direct sum, both vial lookup tables)
8:       end
```

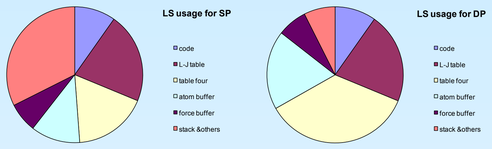
We implemented a simplified version of the kernel that excludes pairlists and bonded forces:

```
1: for each atom i in patch p_k
2:       for each atom j in patch p_l
3:             if atoms i and j are within the cutoff distance
4:                   compute non-bonded forces (L-J potential and PME direct sum, both via lookup tables)
5:       end
6: end
```

SPU NAMD Kernel implementation

Kernel type	*Code*	*L-J table*	*table_four*	*atom buffer*	*force buffer*	*stack & other*
single-precision	25KB	55KB	45KB	30KB	18KB	83KB
double-precision	25KB	55KB	91KB	48KB	18KB	19KB

Table 1: SPE LS memory usage for different kernel types



- Entire patch is loaded into SPE's local storage
- Double-precision case just fits, single-precision still have some space left
- Substantial number of data movement operations are needed

```
vector float * m = (vector float*) pj;
vector float x,y,z,c, delta_x, delta_y, delta_z, r2;

{
     vector float v1, v2, v3, v4, v5, v6, v7, v8, v9;

     v1 = spu_shuffle(*m, *(m+1), ptn1);

     v2 = spu_shuffle(*m, *(m+1), ptn2);
     v3 = spu_shuffle(*(m+2), *(m+3), ptn3);
     v4 = spu_shuffle(*(m+2), *(m+3), ptn4);
     v5 = spu_shuffle(v4, *(m+4), ptn5);
     v6 = spu_shuffle(v4, *(m+4), ptn6);
     v7 = spu_shuffle(v1, v3, ptn7);
     v8 = spu_shuffle(v1, v3, ptn8);
     v9 = spu_shuffle(v2, v3, ptn9);

     x = v7;
     y = spu_shuffle(v5,v8, ptn_y);
     z = spu_shuffle(v5,v2, ptn_z);
     c = spu_shuffle(v9,v6, ptn_c);
   }


   delta_x = spu_sub(ix, x);
   delta_y = spu_sub(iy, y);
   delta_z = spu_sub(iz, z);

   r2 = spu_madd(delta_x, delta_x, r2);
   r2 = spu_madd(delta_y, delta_y, r2);
   r2 = spu_madd(delta_z, delta_z, r2);
    ....
```

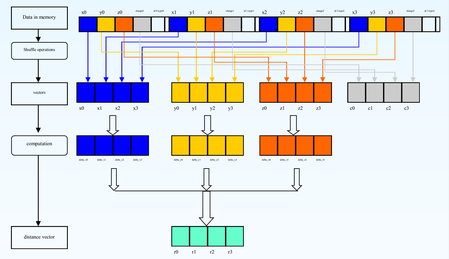


Figure 6: Data movement for distance computation (SP case)

NAMD on Cell: program flow

Different vectorization schemes are applied in order to get best performance:

- **Self-compute**: fill zeros in unused slots
- **Pair-compute**: save enough pairs of atoms, then do calculations

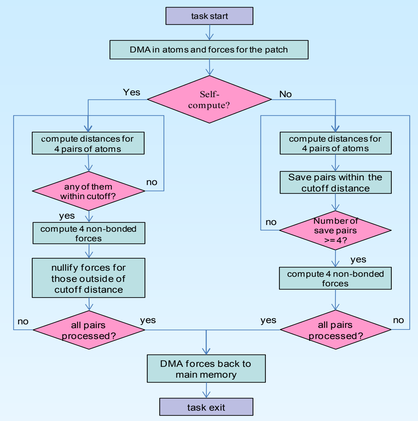


Figure 7: SPE Program flow

PPE runs a task dispatch system:

- Find dependency-free task
- Schedule the task on the idle SPE

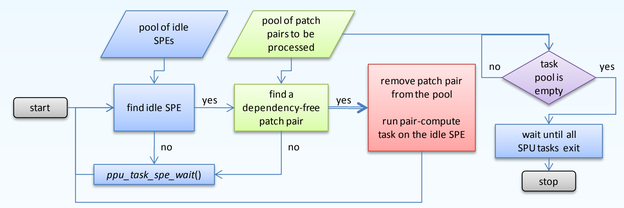


Figure 8: Task dispatch system running on the PPE

Performance

- Distance computation code takes most of the execution time
- Data manipulation time is significant

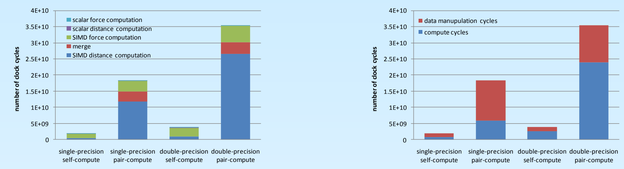


Figure 9: Static analysis of the computing kernels performed with the help of the timing tool, spu_timing

- Perfect scaling on multiple SPEs
- >10x speedup compared with Intel Xeon processor

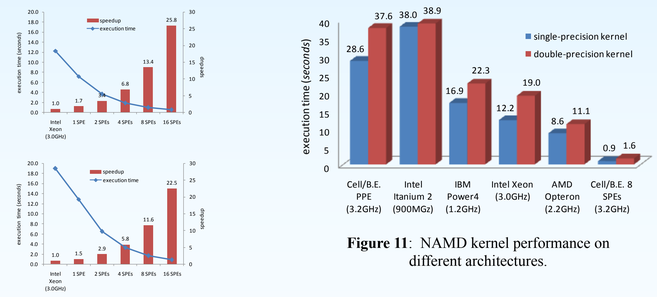


Figure 11: NAMD kernel performance on different architectures.

Figure 10: Scaling and speedup of the force-field kernel implementation as compared to a 3.0 GHz Intel Xeon processor.

Acknowledgements

This work was funded by National Science Foundation grant SCI 05-25308 and by the IBM Linux Technology Center. NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. We are grateful to Dr. James Phillips from the Theoretical and Computational Biophysics Group for his help with NAMD. We acknowledge Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation for the use of Cell Broadband Engine resources that have contributed to this research. We are also thankful to Dr. Paul Woodward from the University of Minnesota for providing access to the Cell blade system. Special thanks to Jeremy Enos from NCSA's Innovative System Lab (ISL) for installing and administrating the Cell/B.E. blade system at NCSA, Dr. John Larson from ISL for useful comments on performance evaluation, and Trish Barker from NCSA's Office of Public Affairs for help in preparing this publication.