Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications

Authors: Vipin Sachdeva, Michael Kistler, Evan Speight and Tzy-Hwa Kathy Tzeng

Presentation by: Keyur Malaviya

Dept of Computer & Information Sciences
University of Delaware

CISC 879 : Software Support for Multicore Architectures
Overview

• **Problem:**
  Genomic data/computational requirements growing
  General-purpose processors cannot handle this

• **Approach:**
  Parallelize & port existing applications to multicore
  Use multicore architecture like Cell (PS3)
  Steps: ➔ Pick applications, Perform profiling
  ➔ Make necessary code changes for porting
  ➔ Publish results

• **Goal:**
  Validate performance gain on PS3
• **Terminology:**
  Gemones ≈ DNA strand ≈ Proteins

• **A typical problem:**
  Machine learning, Prediction, Data mining
  Support vector based, HMM, Clustered

• **Few examples:**
  Determine biological functions of proteins
  Understand biochemical pathways
  Assemble strings to make Genome

• **Approach:**
  **Compare** sequence data with known genomes
Experimental Setup

Two applications selected

1) Sequence alignment: FASTA (sssearch34)
   - Smith-Waterman algorithm \[ O(nm) \]
   - Pairwise alignment of gene sequences
   - Uses dynamic programming algorithm

2) Homology detection: ClustalW (clustalw)
   - Multiple sequence alignment
Sequence alignment basics

- Pairwise alignment: Most commonly performed tasks in bioinformatics

- **To align two sequences:**
  1. Alignment score Matrix (e.g: Blossom, PAM)
  2. Comparing them, assign scores
  3. Insert gaps in one or both sequences
  4. Traceback Dynamic programming table to Produce an optimal score
To align two sequences:

- Alignment score Matrix (e.g: Blossom, PAM)
- Comparing them, assign scores
- Insert gaps in one or both sequences
- Traceback & produce an optimal score

Two sequences:

```
  P1  KQELYLL
  P2  VEYL
```

Aligned Sequences:

```
P1  KQ.EYLL
P2  ..VEYL
```

Scoring table:

<table>
<thead>
<tr>
<th></th>
<th>K</th>
<th>Q</th>
<th>E</th>
<th>Y</th>
<th>L</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>6</td>
<td>-2</td>
<td>-2</td>
<td>-9</td>
<td>-5</td>
<td>-1</td>
</tr>
<tr>
<td>Q</td>
<td>-2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>-2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>-9</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>-5</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
ClustalW basics

• **In three steps:**

  i. All sequences are compared pairwise (Smith-Waterman algorithm)

  ii. Create a hierarchy for alignment (guide tree) by cluster analysis (distance matrix) for each pair

  iii. Progressive add one sequence according to the guide tree to get multiple sequence alignment
Guided tree or Phylogenetic tree

CISC 879 : Software Support for Multicore Architectures
Applications’ characteristics

- Embarrassingly parallel computation
- Small critical \textit{time-consuming} code size
- Regular memory accesses
- Vectorized code
Perform profiling (gprof)

- More than half the execution time is taken by a single function:
  - FASTA: *dropgsw*
  - ClustalW: *forward pass*
  - HMMER: *P7Viterbi*

**Figure:** Execution profile from gprof for three applications & the BLASTapp.
Porting to Cell Architecture

- Altivec: is a PowerPC from AIM alliance

- Altivec SSE implementations:
  - `dropgsw` for FASTA and `P7Viterbi` for HMMER

- IBM Life Sciences:
  - Vectorized kernel for `forward_pass` of ClustalW

- Ported with few modifications to CELL
Porting to Cell Architecture

- Advantage on CELL:
  Used 9 cores: PPU also as a processing element

- Compilation of programs:
  For CELL: XLC v8.1 with -O3
  PowerPC G5: -O3 -mcpu=G5 -mtune=G5
  Opteron & Woodcrest: -O3
FASTA (Smith-waterman) on CELL

- FASTA package includes Altivec-enabled Smith-Waterman
- Smith_waterman_altivec_word kernel was ported with simple changes to CELL
- Altivec APIs: vec_max, vec_sub were written for SPU
- Pairwise alignment of 8 pairs of sequences, using one SPU for each pairwise alignment

**Limitation in current implementation:**
Size of both sequences <= SPU local store (256 KB)
Sequence size <= 2048 characters
Long sequence comparisons:

• **To do genome-wide or long sequence comparisons:**
  - Implement pipelined approach among SPUs
  - Each SPU performs Smith-Waterman alignment for a block, notifies the next SPU through a mailbox message
  - Later SPU uses boundary results of previous SPU for its own block computation

• **Future research:**
  Support of bigger sequences on the Cell
Performance of Smith-Waterman Alignment execution time for different processors:

1: Sequence length = 1024
2: Sequence length = 2048
**Pairwise Alignment**

- **pairalign function**: All-to-all pairwise comparisons for ‘n’ sequences performs \( n(n-1)/2 \) alignments

- Takes 60%-80% of the execution time

- **pairalign** function is made of 4 functions

- 1) **Forward_pass** computes the maximum score and is the most time-consuming step of **pairalign**
• **Input to ClustalW:**
  - ‘n’ sequences in a query sequence file
  - n(n−1)/2 computations

• **Mfc DMA in/out from 16-byte boundaries:**
  Pack all sequences in a single array
  Each sequence begin at a multiple of 16-bytes
• PPU creates threads, passes max sequence size

• SPUs wait for PPU to send a message to pull in the context data & begin computation

• **Work distribution Round-robin strategy:**

  o Each SPU is assigned a number from 0 to 7

  o if ‘i mod 8=k’ SPU ‘k’ compares seq no. ‘i’ against all sequences ‘i+1’ to ‘n’

    \[
    \begin{align*}
    i=9 & \Rightarrow k=0 \\
    i=15 & \Rightarrow k=7
    \end{align*}
    \]

    [ so on…]
Issues in porting ClustalW and bottlenecks

- IBM Life Sciences vectorized version of `forward_pass`
- Altivec APIs: `vec_max`, `vec_adds` written for SPUs
- SPUs don’t support 16-bit: Altivec used vector status & control register to detect overflow on it
- SPU use of 32-bit (`int`) lowers vector computing efficiency (only 4 values can be packed in a vector)
- SPU has only vector registers: Reading Alignment matrix score – a scalar operation suffers on SPU
Issues in porting ClustalW and bottlenecks

- **SPUs only have static branch prediction:**
  - Fails on a branch with multiple conditions or
  - More loop variables handling boundary cases
  - Such branches are difficult to predict for the SPU

- **Solution:**
  - Make branch depend on a single loop variable
  - Break inner alignment loop into several loops
  - Explicit handling of boundary cases

- **2X performance gains**
Code changes behavior

Improvement of performance of ClustalW alignment function with different code changes.

Best implementation: Using integer datatypes with no branches
Performance of \textit{forward\_pass} (ClustalW)

Comparison of Cell Performance with other processors for only alignment function with simple round-robin strategy

Two inputs from BioPerf suite:
1) 1290.seq has 66 sequences of average length 1082
2) 6000.seq has 318 sequences of average length 1043

NOTE: Opteron and the Woodcrest performance is non-vectorized

CISC 879 : Software Support for Multicore Architectures
Performance of ClustalW

Comparison of Cell Performance with other processors for total time of execution

Two inputs from BioPerf suite:
1) 1290.seq has 66 sequences of average length 1082
2) 6000.seq has 318 sequences of average length 1043

NOTE: Opteron and the Woodcrest performance is non-vectorized
Computing the final alignment

- Performance gain in Forward_pass
- Final step executing on PPU is much slower compared to other superscalar processors

**Two approaches to this problem:**
Execute more code on the SPUs, or
Use Cell as an accelerator, along with a superscalar processor

**Future work:**
Increasing PPU performance (RoadRunner project is exploring such hybrid architectures)
PPU penalty:

Cell performance is marginally better in overall execution time due to performance of the PPU.

**Forward_pass**

- Cell (8 SPUs): 7.5 times faster than Power G5
- Overall: Cell (8 SPUs) 1.2 times faster than Power G5

*CISC 879 : Software Support for Multicore Architectures*
Current progress in porting

- Many bioinformatics applications are being ported to the Cell processor
- **2500** playstations used to parallelize gene-finding and sequence alignment softwares
- Protein folding on distributed computing and GPUs
- Charm++ runtime system, used for NAMD simulations
- FASTA, ClustalW, and HMMER
- …
Conclusions & Future Work

- Cell’s total power consumption < half of a superscalar processor
- Cell is price & power-efficient platform for future bioinformatics computing

**Future Work:**
- Removing limitations & increasing optimization
- Porting HMMER to Cell processor
- Handling size that exceeds 256 KB
- Use of partitioning input among 8 SPUs
- Porting protein docking, RNA interference, medical imaging and few more applications
QUESTIONS