Improving CUDASW++, a parallelization of Smith-Waterman for CUDA enabled devices

Conference Paper · June 2011

DOI: 10.1109/IPDPS.2011.182 · Source: IEEE Xplore

Doug Hains
Colorado State University
8 PUBLICATIONS **128** CITATIONS
SEE PROFILE

A. P. Wim Böhm
Colorado State University
109 PUBLICATIONS **1,368** CITATIONS
SEE PROFILE

Sanjay V. Rajopadhye
Colorado State University
167 PUBLICATIONS **2,080** CITATIONS
SEE PROFILE

Improving CUDASW++, a Parallelization of Smith-Waterman for CUDA Enabled Devices

Doug Hains, Zach Cashero, Mark Ottenberg,
Wim Bohm and Sanjay Rajopadhye

April 28, 2011

Database Sequence Alignment

- One or more *Query* sequences
- Compared to a database of sequences
- Want to know only the maximum similarity scores
- Performance measured in Giga-cell updates per seconds (GCUPS)
- CUDASW++ uses Smith-Waterman using affine gap penalties
  - ρ gap open penalty
  - σ gap extension penalty
- We identified a bottleneck in CUDASW++ and developed a wavefront tiling strategy.
- Reduces the effect of the bottleneck and increases the overall performance of the algorithm.

Smith-Waterman

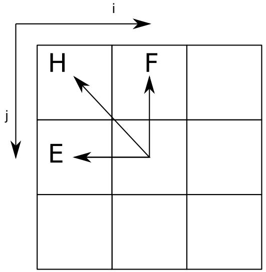
- Using an affine gap penalty we have the following dependencies:

$$E_{i,j} = \max \left\{ \begin{array}{l} E_{i,j-1} - \sigma \\ H_{i,j-1} - \rho \end{array} \right\}$$

$$F_{i,j} = \max \left\{ \begin{array}{l} F_{i-1,j} - \sigma \\ H_{i-1,j} - \rho \end{array} \right\}$$

$$H_{i,j} = \max \left\{ \begin{array}{l} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + w(q_i, d_j) \end{array} \right\}$$

  - $H_{i,0} = H_{0,j} = E_{i,0} = F_{0,j} = 0$

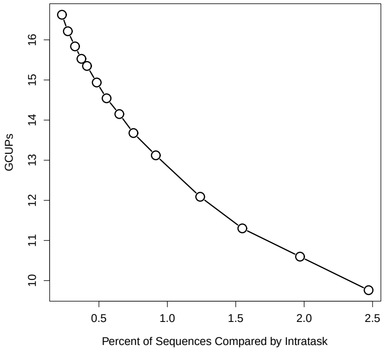


Overview of CUDASW++

- Two different tasks (kernels): Inter-task and Intra-task
- Tasks chosen on basis of database sequence length
- Default threshold of 3072

- Inter-task
  - Database seq. length < 3072
  - **One thread** for each sequence alignment
  - Optimizations to reduce global memory traffic
  - Dependencies stored locally
- Intra-task
  - Database seq. length ≥ 3072
  - **One thread block** for each sequence alignment
  - High amount of global memory traffic
  - Dependencies stored globally

Intra-task is a Bottleneck



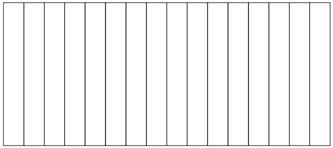
- Ran CUDASW++ on UniProtKB/Swiss-Prot database (500,000+ sequences), varied the threshold so more sequences are run with Intra-task
- By increasing the use of intra-task slightly, the performance goes down considerably
- So, why is intra-task needed?

Inter-task memory structure: Motivation for intra-task

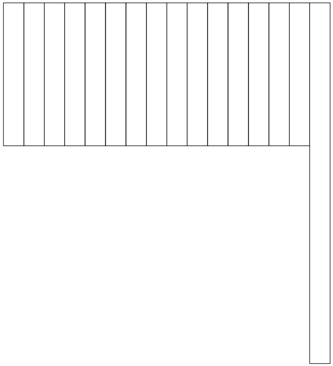
- First, determine the number of database sequences compared to a query sequence per kernel call. (Based on device)
- Sequences are sorted by length and packed into groups in global memory.
- Size of the group is the total number of threads to be launched by the inter-task kernel. (1:1 thread to database sequence ratio)
- Sequentially launch kernels until all groups are compared.
- **Cannot make the next kernel call until all threads from previosu kernel call are done.**

Load balancing in Inter-task

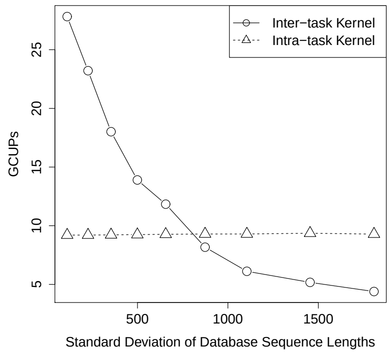
- Best Case:



- Worst Case:

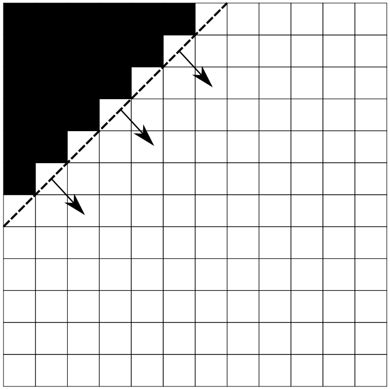


Intra-task is not Affected



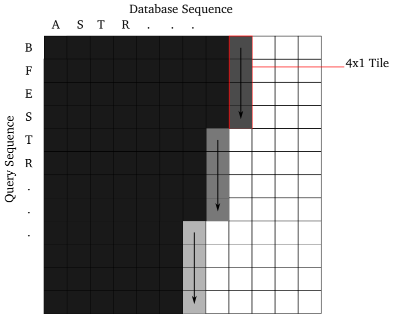
- Generated multiple databases of random sequences.
- Used a log-normal distribution to shift distribution of sequence lengths from best-case to worst-case.
- Intra-task is faster when distribution of lengths is uneven
- Can we improve it?

Original Intra-task



- Table is computed by diagonals which are stored in Global memory.
- Requires a number of global read/writes proportional to the length of the diagonal.
- Idle threads when diagonal length is not a multiple of thread block size.

Improving Intra-Task



- Use tiling to:
  - Reduce global memory read/writes
  - Reduce idle threads (Threads only idle during pipeline fill/empty)

Initial Results

Used a randomly generated database with all sequence lengths the threshold.

- Original CUDASW++ : 1.5 GcupS

Initial Results

Used a randomly generated database with all sequence lengths the threshold.

- Original CUDASW++ : 1.5 GcupS
- "Improved" CUDASW++ : 1.5 GcupS

Local Memory Usage - Shallow Copy

- Original CUDASW++: 0 bytes lmem
- Improved CUDASW++: 64 bytes lmem

```
int *tmp;
int h1[4], h2[4];
while(...)  { /* Loop over tiles */
        ...
        tmp = h1;
        h1 = h2;
        h2 = tmp;
}
```

Local Memory Usage - Texture Memory

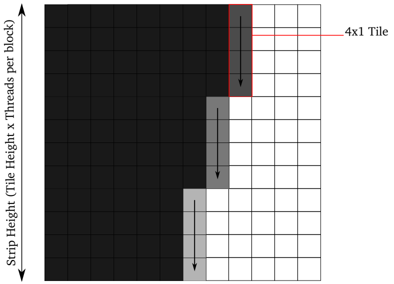
```
#define TILE_WIDTH 4
int tmpH[TILE_WIDTH];

for(k = 0; k < TILE_WIDTH; k++) {  /* Loop over columns */
        db_char = tex2d(...); /* read db symbol */
        ...
        tmpH[k] = ...
}
```

- Loops containing texture memory calls cannot be unrolled

Use the Query Profile

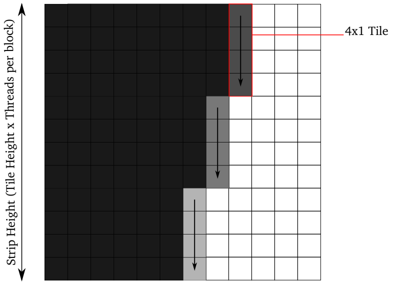
- Query Profile (from Rognes, also used in Inter-task)
  - Similiarty scores (w) are precomputed, storing 4 scores in 4 bytes.
- Before Query Profile
  - Each query symbol read from global memory.
  - Similarity scores read from lookup table in shared memory.
- After Query Profile
  - Retrieves four similarity scores directly per global read.
  - No shared memory reads required.

Thread Block Size and Tile Height



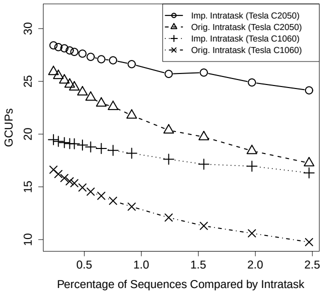
- Strip height: tile height $\times$ thread block size
- Here, strip height $= 4 \times 3 = 12$
- If query sequence $>$ strip height, intermediate results stored in global memory.
- Found strip height of 512 to be optimal on C1060 and 1024 on C2050.
- Variations on thread block size/tile height with equal strip heights had similar performance.

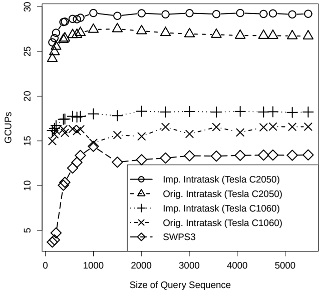
Tile Width



- Tile Width does not change the number of global/shared memory accesses.
- Tile Width does increase the pipeline latency.
- Tile Width of one is optimal.

Results



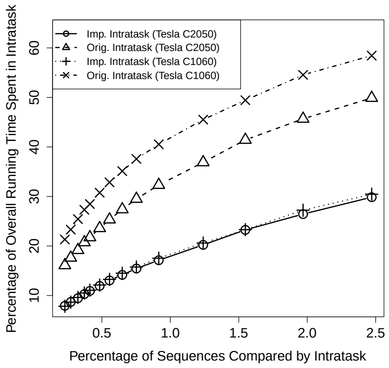


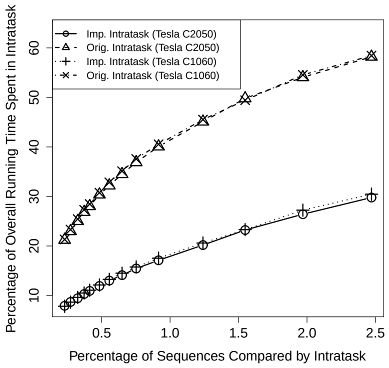
Kernel	Global Memory Transactions Query Len. 567	Global Memory Transactions Query Len. 5478
Imp. Kernel	13,828	4,233,197
Orig. Kernel	28,345,473	291,179,739

Table: Number of total global memory accesses performed by both kernels on queries of two different sizes against the Swissprot database.

Is Global Memory Caching (Fermi) Comparable?

- Cache turned on for C2050
- Cache turned off for C2050





- Although we show an increase in performance of the original CUDASW++ due to caching, it is not as good as explicitly using shared memory.

Conclusions and Future Work

- Conclusions
  - Intra-task was a bottleneck in CUDASW++
  - It is necessary due to load balancing issues with inter-task
  - Our wavefront tiling method improves performance on both C2050 and C1060 GPUs
- Future work
  - Integrate our improvements with official version of CUDASW++
  - Explore other ideas for performance tuning