Autotuning Wavefront Applications for Multicore Multi-GPU Hybrid Architectures

University of Edinburgh

Siddharth Mohanty
Murray Cole
Agenda

- Wavefront Pattern (1:00)
- Wavefront Applications (0:30)
- Implementation Strategy + trade-offs (4:30)
- Experimental Programme (1:30)
- Platform And Parameters (1:00)
- Exhaustive Search Results (2:00)
- ESR : Best Points Performance (1:00)
- ESR : Best Points Sensitivity (1:00)
- Autotuning Model (1:00)
- Autotuning Results (1:30)
- Q&A (4:00)
Wavefront Pattern

Wavefront Applications

- Nash Equilibrium: A game-theoretic problem in economics, characterized by small instances but a very computationally demanding kernel. The internal granularity parameter controls the iteration count of a nested loop.

- Biological Sequence Comparison: A string alignment problem from Bioinformatics, characterized by very large instances and very fine-grained kernels, varying with detailed comparisons made.

A matrix $H$ is built as follows:

$H(i, 0) = 0, \ 0 \leq i \leq m$
$H(0, j) = 0, \ 0 \leq j \leq n$

if $a_i = b_j$ then $w(a_i, b_j) = w(\text{match})$ or if $a_i \neq b_j$ then $w(a_i, b_j) = w(\text{mismatch})$

$$H(i, j) = \max \left\{ \begin{array}{ccc} 0 & \text{Match/Mismatch} \\ H(i-1, j-1) + w(a_i, b_j) & \text{Deletion} \\ H(i-1, j) + w(a_i, -) & \text{Insertion} \\ H(i, j-1) + w(-, b_j) & \end{array} \right\}, \ 1 \leq i \leq m, 1 \leq j \leq n$$

Where:

- $a, b =$ Strings over the Alphabet $\Sigma$
- $m = \text{length}(a)$
- $n = \text{length}(b)$
- $H(i, j)$ - is the maximum Similarity-Score between a suffix of $a[1..i]$ and a suffix of $b[1..j]$
- $w(c, d), \ c, d \in \Sigma \cup \{\text{-}\}$, '-' is the gap-scoring scheme

Implementation Strategy

Dual GPU MultiCore Wavefront Framework

20x20 problem grid, 4x4 cpu-tile, 1x1 gpu-tile, 5 diagonals

- Computed in CPU
- Computed in GPU

CPU

GPU

GPU 1

GPU 1 HALO

GPU 2

GPU 2 HALO

(4:30)
Experimental Programme
## Platforms and Parameters

<table>
<thead>
<tr>
<th>Parameter Description</th>
<th>Parameter Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dim$ width of the array</td>
<td>$dim$ 500 to 3100</td>
</tr>
<tr>
<td>$tsize$ granularity of the element computation</td>
<td>$tsize$ 10 to 12000</td>
</tr>
<tr>
<td>$dsize$ element data size</td>
<td>$dsize$ 1, 3, 5</td>
</tr>
<tr>
<td>$cpu$-tile side length of the square tiles for CPU tiling</td>
<td>$cpu$-tile 1, 2, 4, 8, 10</td>
</tr>
<tr>
<td>$band$ number of diagonals on each side of the main</td>
<td>$band$ -1 to $2^{*}dim-1$</td>
</tr>
<tr>
<td>diagonal, to be computed on the GPU</td>
<td></td>
</tr>
<tr>
<td>$gpu$-count number of GPU devices to use</td>
<td>$gpu$-count 0, 1, 2</td>
</tr>
<tr>
<td>$gpu$-tile the GPU equivalent of CPU tiling</td>
<td>$gpu$-tile -1 to 0.5*(length of first offloaded diagonal)</td>
</tr>
<tr>
<td>$halo$ size of the halo for dual GPUs</td>
<td></td>
</tr>
</tbody>
</table>

### Table 1: Input Parameters

### Table 3: Parameter Ranges

<table>
<thead>
<tr>
<th>System</th>
<th>Freq (MHz)</th>
<th>Cores (HT)</th>
<th>Mem (GB)</th>
<th>GPU Freq (MHz)</th>
<th>CU Mem (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i3-540</td>
<td>1200</td>
<td>4</td>
<td>4</td>
<td>GTX 480</td>
<td>1401</td>
</tr>
<tr>
<td>i7-2600K</td>
<td>1600</td>
<td>8</td>
<td>8</td>
<td>4x(GTX 590)</td>
<td>1215</td>
</tr>
<tr>
<td>i7-3820</td>
<td>3601</td>
<td>8</td>
<td>16</td>
<td>Tesla C2070, C2075</td>
<td>1147</td>
</tr>
</tbody>
</table>

### Table 2: Tunable Parameters

### Table 4: Experimental Systems
Figure: Heatmaps illustrate the *band* and *halo* values at the best performing points from our exhaustive search across three systems and element size of 16 bytes (*dsize*=1; 1 float and 2 ints) and 48 bytes (*dsize*=5; 5 floats and 2 ints). The i3 system is a single GPU system, hence no halo heat map is shown. In all maps the x-axis is *tsize*, indicating kernel task granularity and the y-axis is *dim*, indicating problem size.
Figure: Average case comparison for the Synthetic Application. The x-axis is \textit{dim-tsize}, indicating groups of problem sizes whose kernel task granularity varies from 10 to 12K and the y-axis is \textit{rtime}, indicating actual runtime. Best is the best exhaustive \textit{rtime} (\textit{ber}), AVG is the average \textit{rtime} from all configurations, S.D. is the standard deviation from average. \textit{ds}ized refers to the number of floats in our synthetic data structure containing 2 int variables. Total element size = 16 bytes (\textit{ds}ize=1; 1 float and 2 ints) and 48 bytes (\textit{ds}ize=5).
ESR : Best Points Sensitivity (1:00)

Figure : Violin plots showing dispersion of all configurations. The best points are at the base and the white spots are the medians. The x-axis is tsize, indicating kernel task granularity and the y-axis is rtime, indicating actual execution time.
Figure : i7-2600K system : The M5 pruned model tree for predicting halo values with one linear model (out of 22) shown. As seen, halo depends on band and cpu-tile values, apart from the input parameters of task granularity and data granularity.
Autotuning Results

Auto-tuning Performance (Averaged)

Nash Application

NASH APP
4x(i3-540) | GeForce GTX 480

8x(i7-3820) | Tesla C2070, Tesla C2075

8x(i7-2600K) | 4x(geForce GTX 590)
Thank You
Appendix : Tuning Challenges

- Problem size ($\text{dim}$) large enough to justify parallel computation in GPU (smaller sized problems can be computed quicker in the faster CPU cores)
- Granularity of task ($\text{tsize}$) high enough for computation to dominate over the cost of starting a GPU and the communication overhead of transferring data between GPU and CPU.
- Communication cost increases with increase in data ($\text{dsize}$) being transferred
- Dual GPUs have the additional overhead of exchanging neighbouring data between themselves every few iterations ($\text{halo}$ swapping).
- Halo swaps will decrease with increase in halo size but this has to be traded against redundant computation, which starts affecting performance with increase in granularity of task
- GPU tiling ($\text{gpu-tile}$) leads to reduction in the number of kernel calls but this has to be traded against the additional cost of synchronizing work items within each work group.
- When computation dominates over communication anyway, time spent in kernel calls no longer matters and gpu tiling may prove to be counter productive
- The type of system affects the performance :
  - fast GPU coupled to a slow CPU means data will mostly be offloaded to the GPU, meaning more diagonals in the GPU ($\text{band}$ sizes) with CPU tiling having negligible effect.
  - fast GPU + fast CPU would similarly mean lower band sizes
Appendix : Framework Interface

```c
struct params {
    int xpos;
    ...
    double v1;
    ...
};

class wavefronttask< params* >(params* params) {
    public:
        void computeKernel(params* x, params* y, params* d) const {
            struct params* r;
            r = (params*)malloc(sizeof(*r));
            int X,Y;
            //get the values: x=left, y=bottom, d=diagonal
            X=d->xpos; Y=d->ypos;
            ...
            //do the computation with values x,y,d
            nash1 = 4 - 12*pow(q1,2) + ... - 2*d->v1 - ...+2*x->v1 + 2*q2*x->v1...
            //return results
            r->xpos=X;r->ypos=Y;r->v1=bestnash1;
            r->v2=bestnash2;r->q1=bestq1;r->q2=bestq2;
            return r;
        }
    }

class wavefronttask< params* >
    main()
    ...
    wavefronttask* wdoobj = new ...
    e=North_West_NorthWest;
    sl->set_dependency(e);
}
```

**Figure** : Using the Wavefront Skeleton- defining custom data type and overriding base class

```c
int F[MAX_LEN+1][MAX_LEN+1];
void SerialLCS(const char* x, size_t xlen, const char* y, size_t ylen) {
    for (size_t i=1; i<xlen; ++i) {
        for (size_t j=1; j<ylen; ++j) {
            int F[i][j] = (x[i-1]==y[j-1]) ? F[i-1][j-1] + 1 : max(F[i][j-1], F[i-1][j]);
        }
    }
}
```

**Figure** : The sequential string comparison kernel with North West dependency
Appendix: TBB/Omp/baseline vs skeleton
Appendix: Previous Autotuning Performance

- Synthetic Application – note varying colour key

---

**Synthetic Application (i3-540): Universal Tuner**

- **rtime (%) Improvement**: Autotuning over Exhaustive
- **band (%) Difference**: Autotuning over Exhaustive

(a) Linear Regression

(b) SVM Regression
Appendix : Previous Summarised Results

- Overall Average Performance

Optimal speed-up found during exhaustive search, and percentage of this obtained automatically by our autotuning techniques. Figures are averaged across the whole application set.

<table>
<thead>
<tr>
<th>System</th>
<th>Optimal Speedup</th>
<th>Tuner</th>
<th>Model</th>
<th>% of Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>i3-540</td>
<td>6.34x</td>
<td>Universal</td>
<td>SVM</td>
<td>87.18%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>85.80%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class Specific</td>
<td>SVM</td>
<td>94.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>95.4%</td>
</tr>
<tr>
<td>i7-990</td>
<td>7.13x</td>
<td>Universal</td>
<td>SVM</td>
<td>40.18%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>57.61%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class Specific</td>
<td>SVM</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>89.5%</td>
</tr>
<tr>
<td>i7-3280</td>
<td>37x</td>
<td>Universal</td>
<td>SVM</td>
<td>39.07%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>48.23%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class Specific</td>
<td>SVM</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>91.7%</td>
</tr>
</tbody>
</table>