Autotuning Wavefront Applications for Multicore Multi-GPU Hybrid Architectures

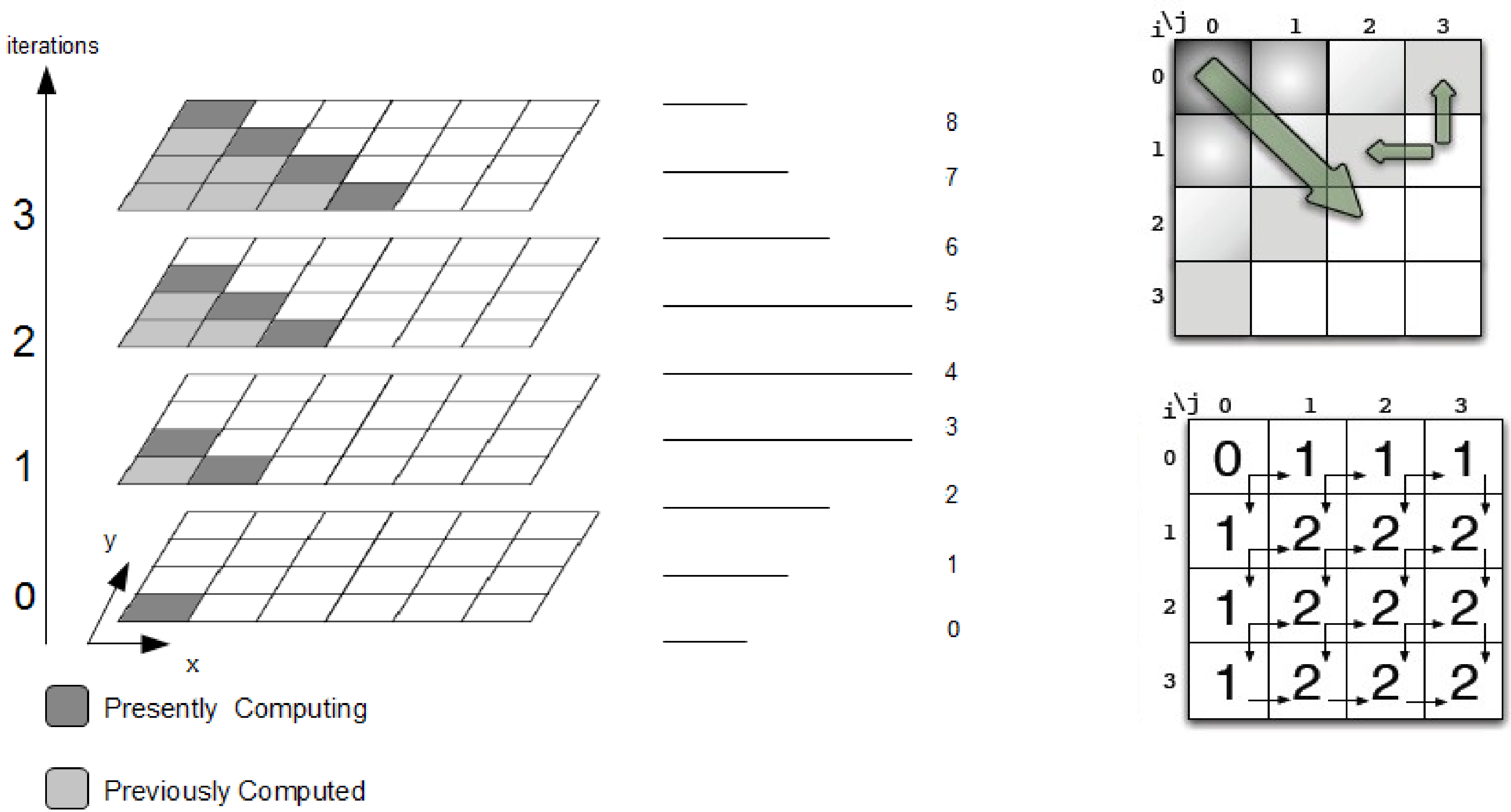
University of Edinburgh

Siddharth Mohanty
Murray Cole

Agenda (1:00)

- 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 (0:30)



(a) (b) (c)

(c)-Dios, A.J et al."Evaluation of the Task Programming Model in the Parallelization of Wavefront Problems," (HPCC), 2010, IEEE

Wavefront Applications (0:30)

- 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:

$H(i,0) = 0,\ 0 \le i \le m$

$H(0,j) = 0,\ 0 \le j \le n$

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

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

Where:

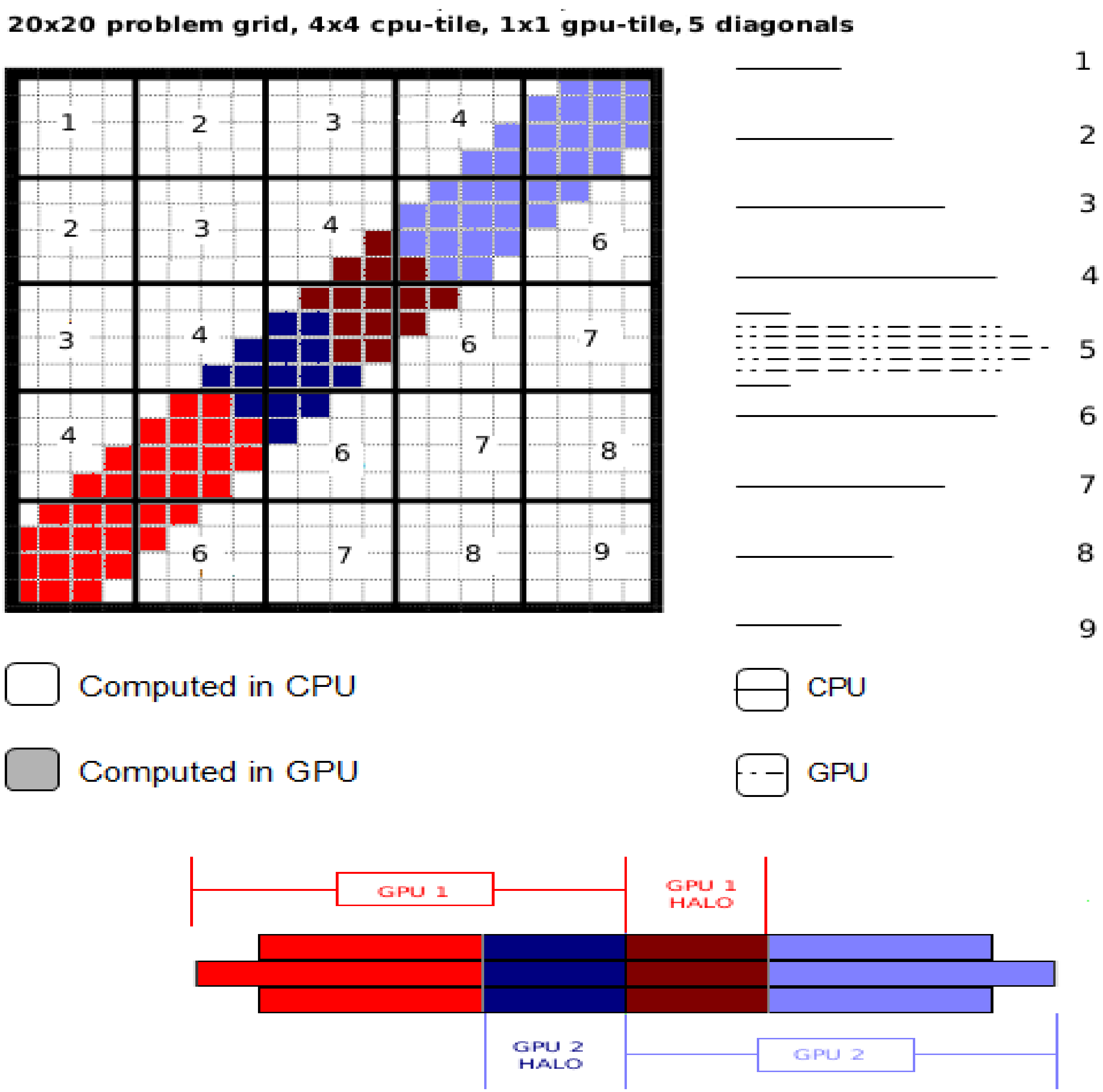
- a, b = Strings over the Alphabet Σ
- $m = \text{length}(a)$
- $n = \text{length}(b)$
- $H(i,j)$ – is the maximum Similarity-Score between a suffix of a[1...i] and a suffix of b[1...j]
- $w(c,d),\ c,d \in \Sigma \cup \{'-'\}$, '–' is the gap-scoring scheme

(a)

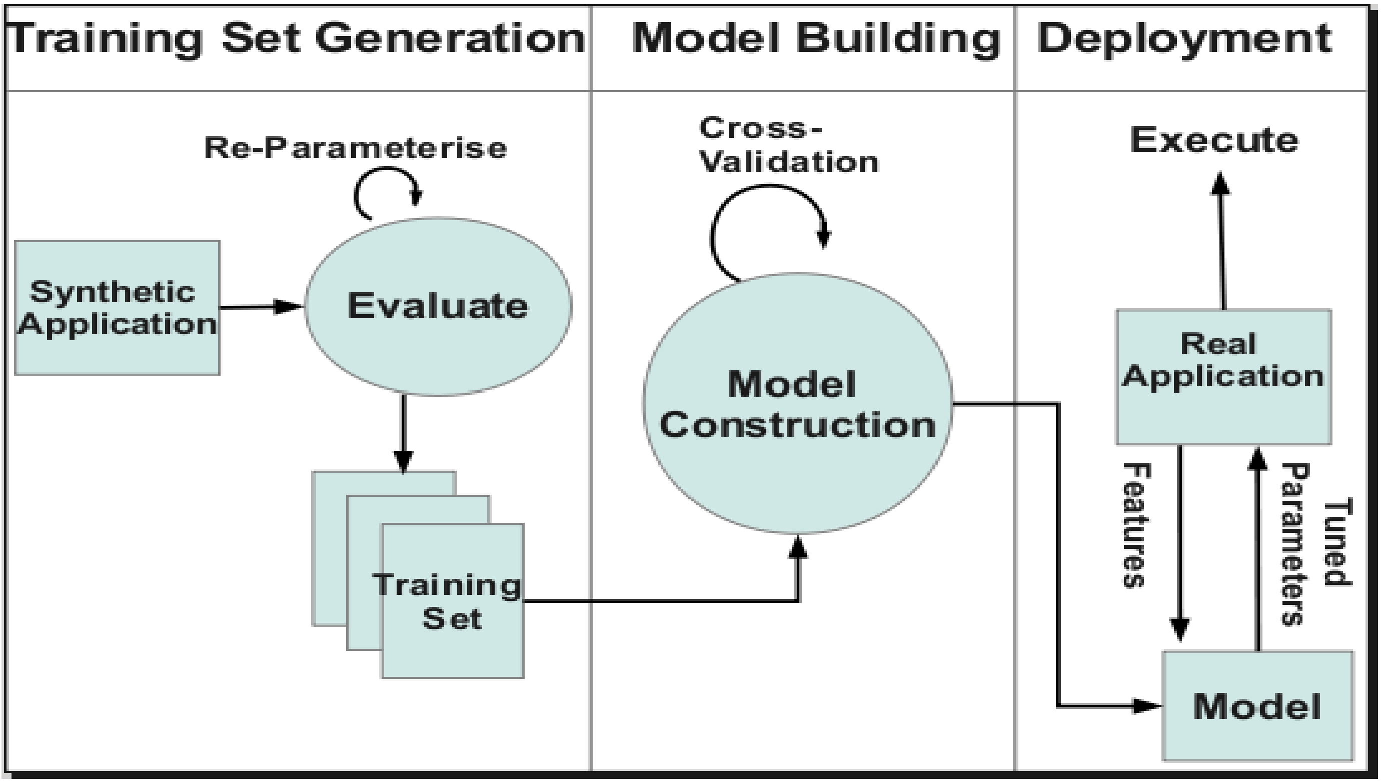
(a)- http://en.wikipedia.org/wiki/SmithWaterman_algorithm

Implementation Strategy (4:30)

Dual GPU MultiCore Wavefront Framework



Experimental Programme (1:30)



Platforms and Parameters (0:30)

Parameter	Description
dim	width of the array
tsize	granularity of the element computation
dsize	element data size

Table 1: Input Parameters

Parameter	Description
cpu-tile	side length of the square tiles for CPU tiling
band	number of diagonals on each side of the main diagonal, to be computed on the GPU
gpu-count	number of GPU devices to use
gpu-tile	the GPU equivalent of CPU tiling
halo	size of the halo for dual GPUs

Table 2: Tunable Parameters

Parameter	Range
dim	500 to 3100
tsize	10 to 12000
dsize	1, 3, 5
cpu-tile	1, 2, 4, 8, 10
band	-1 to 2**dim*-1
gpu-count	0, 1, 2
halo	-1 to 0.5*(length of first offloaded diagonal)
gpu-tile	1, 4, 8, 11, 16, 21, 25

Table 3: Parameter Ranges

System	Freq (Mhz)	Cores (HT)	Mem (GB)	GPU	Freq (Mhz)	CU	Mem (GB)
i3-540	1200	4	4	GTX 480	1401	15	1.6
i7-2600K	1600	8	8	4x(GTX 590)	1215	16	1.6
i7-3820	3601	8	16	Tesla C2070, C2075	1147	14	6.4

Table 4: Experimental Systems

Exhaustive Search Results (ESR) (2:00)

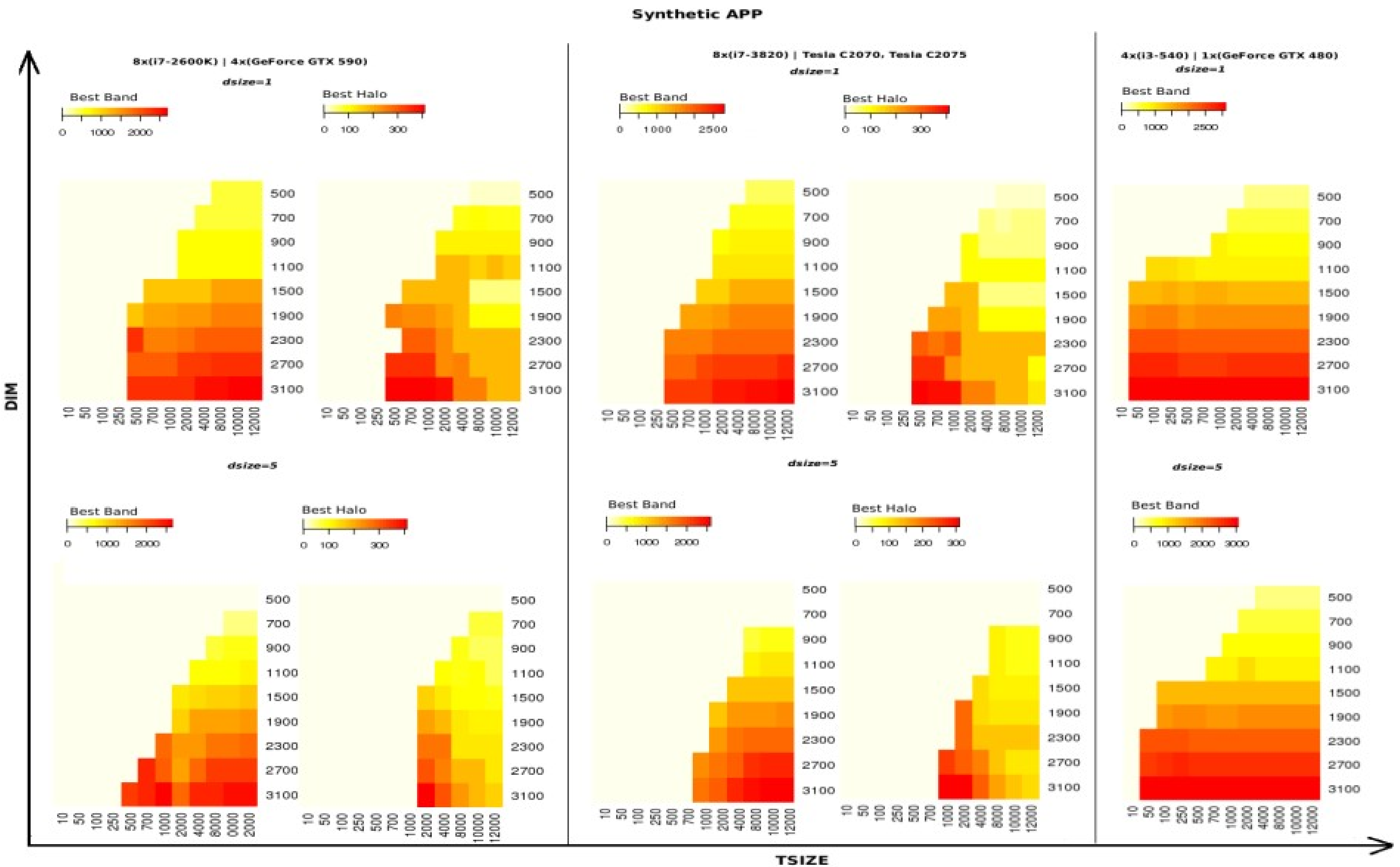


Figure : Heatmaps illustrate the *band* and *halo* values at the best performing points from our exhaustive search across three systems and element size of 16 bytes (*dsize*=1; 1 float and 2 ints) and 48 bytes (*dsize*=5; 5 floats and 2 ints). The i3 system is a single GPU system, hence no halo heat map is shown. In all maps the x-axis is *tsize*, indicating kernel task granularity and the y-axis is *dim*, indicating problem size.

ESR : Best Point Performance (1:00)

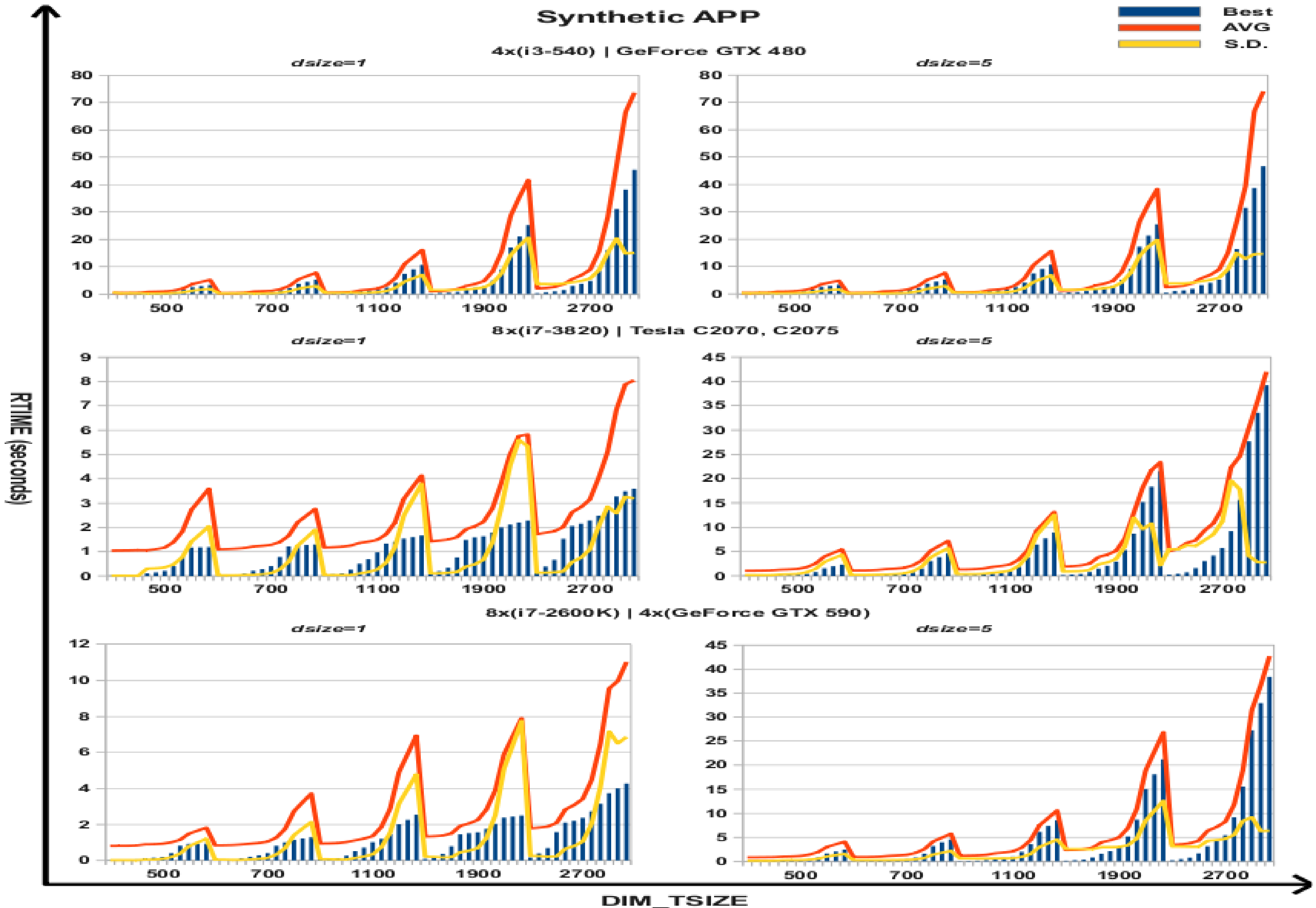


Figure : Average case comparison for the Synthetic Application. The x-axis is *dim-tsize*, indicating groups of problem sizes whose kernel task granularity varies from 10 to 12K and the y-axis is *rtime*, indicating actual runtime. Best is the best exhaustive *rtime* (*ber*), AVG is the average *rtime* from all configurations, S.D. is the standard deviation from average. *dsize* refers to the number of floats in our synthetic data structure containing 2 int variables. Total element size = 16 bytes (*dsize*=1; 1 float and 2 ints) and 48 bytes (*dsize*=5)

ESR : Best Points Sensitivity (1:00)

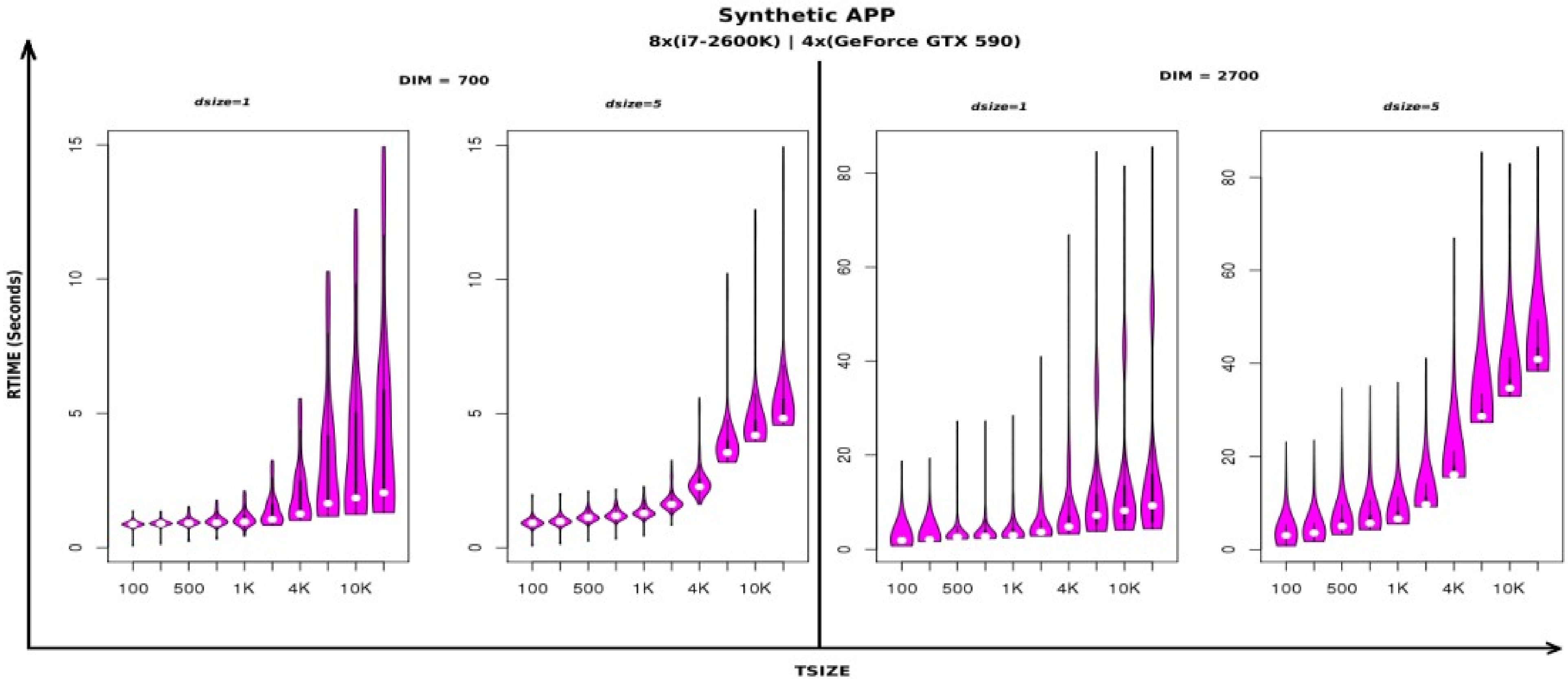


Figure : Violin plots showing dispersion of all configurations. The best points are at the base and the white spots are the medians. The x-axis is *tsize*, indicating kernel task granularity and the y-axis is *rtime*, indicating actual execution time.

Autotuning : Model (1:00)

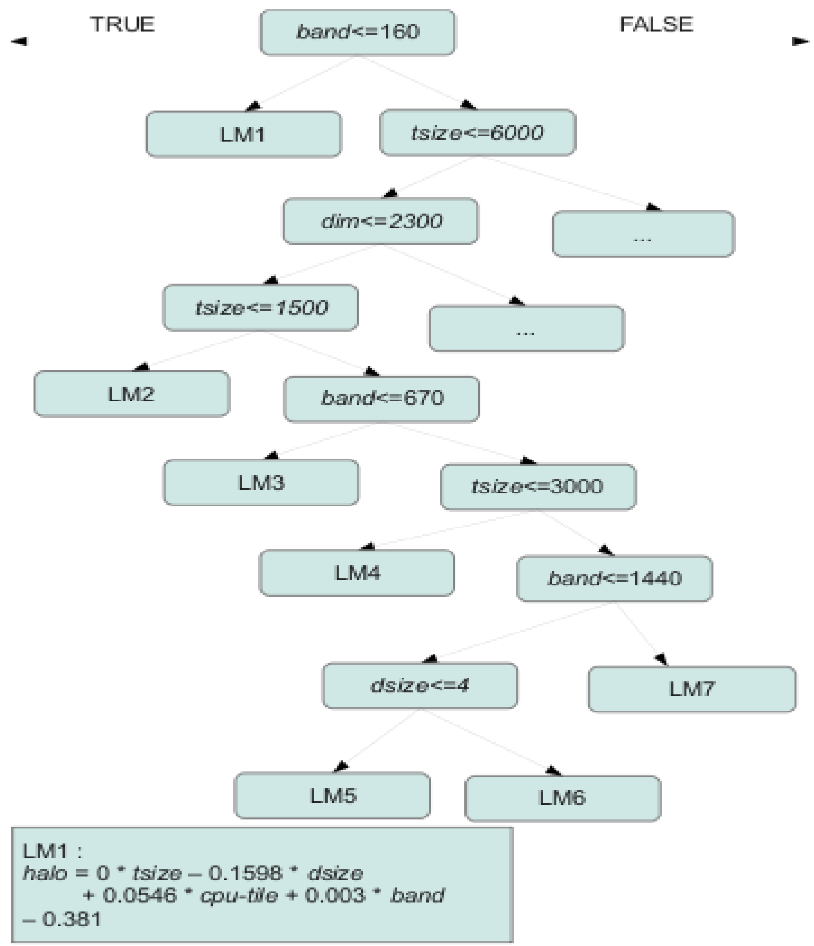
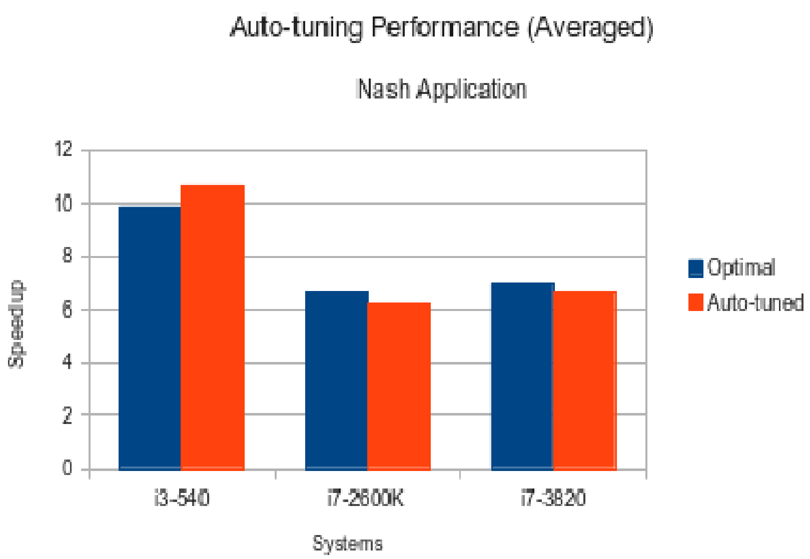
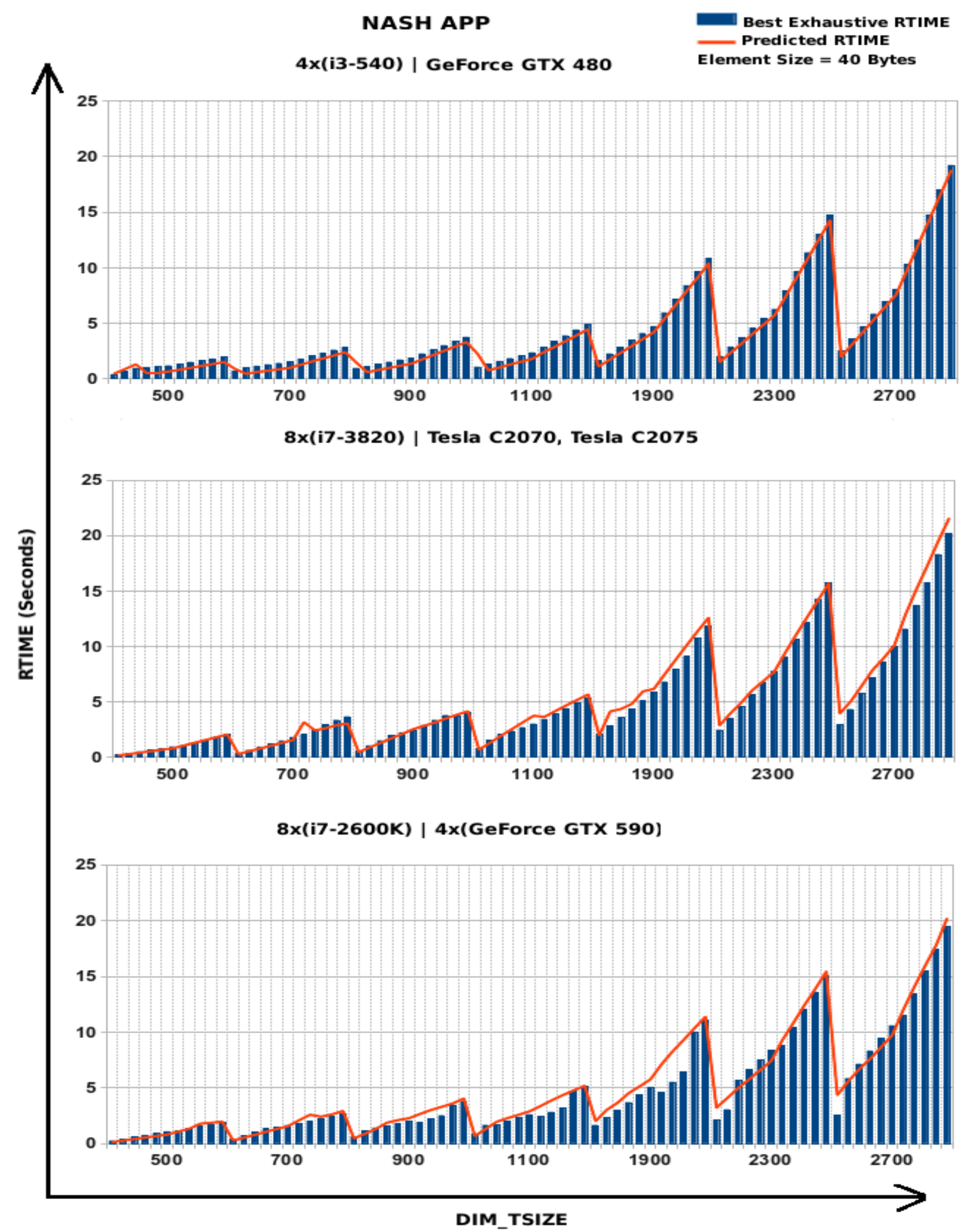


Figure : i7-2600K system : The M5 pruned model tree for predicting *halo* values with one linear model (out of 22) shown . As seen, *halo* depends on *band* and *cpu-tile* values, apart from the input parameters of task granularity and data granularity.

Autotuning Results (1:30)





Thank You

Appendix :Tuning Challenges

- Problem size (***dim***) large enough to justify parallel computation in GPU (smaller sized problems can be computed quicker in the faster CPU cores)
- Granularity of task (***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 (***dsize***) being transferred
- Dual GPUs have the additional overhead of exchanging neighbouring data between themselves every few iterations (***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 (***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 (***band*** sizes) with CPU tiling having negligible effect.

  - fast GPU + fast CPU would similarly mean lower band sizes

Appendix : Framework Interface

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

class computederived:public computematrix< params* >{

    /*A user implementation of how to read input*/
    void params_load( const char *path,int limitM,int limitN){
        ...
         p = (params*)malloc(sizeof(*p));
         fscanf(file,"%d %d %lf %lf %lf %lf",&p->xpos,&p->ypos,&p->v1,
                 &p->v2,&p->q1,&p->q2);
        ...
    }
    /* Overriding the create method*/
     void createcomputematrix(int M,int N){
         ...
         params_load("example.input.data",M,N);
         gettimeofday(&end, NULL);

      }
}
```

```
class wavetaskderived : public wavefronttask< params* >{
public:
     params* 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 from 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;
    }

main()
...
wavetaskderived *wdobj = new ...
e=North_West_NorthWest;
s1->set_dependency(e);
```

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

```
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 )
            F[i][j] = x[i-1]==y[j-1] ? F[i-1][j-1]+1
                        :max(F[i][j-1],F[i-1][j]);
}
```

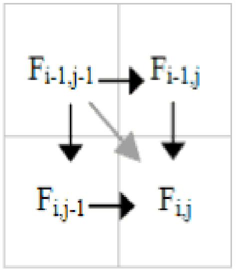
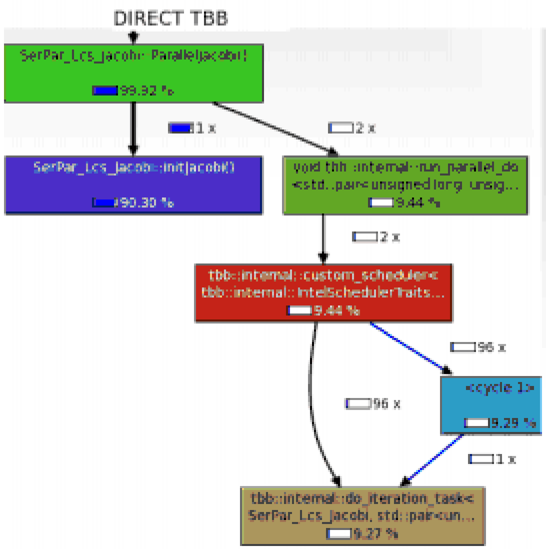
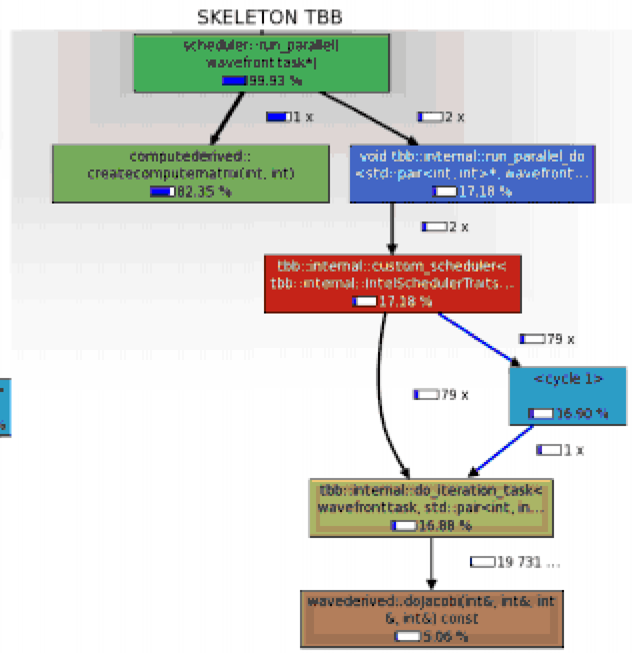


Figure : The sequential string comparison kernel with North West dependency

•Appendix : TBB/Omp/baseline vs skeleton





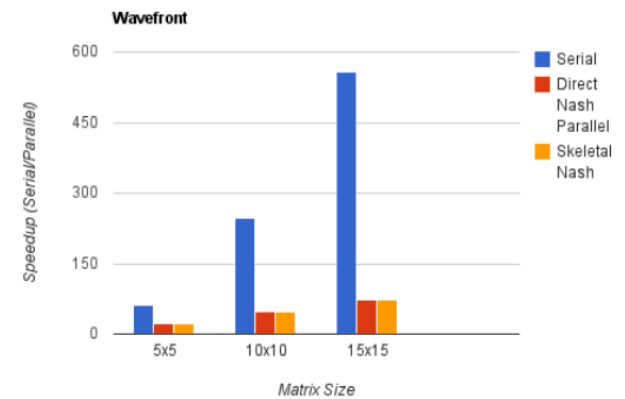
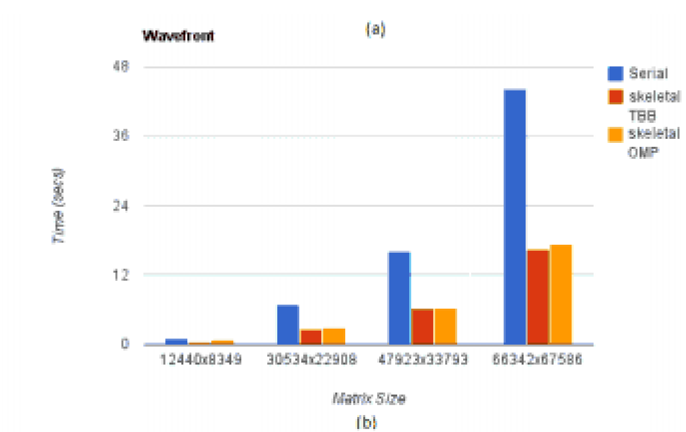
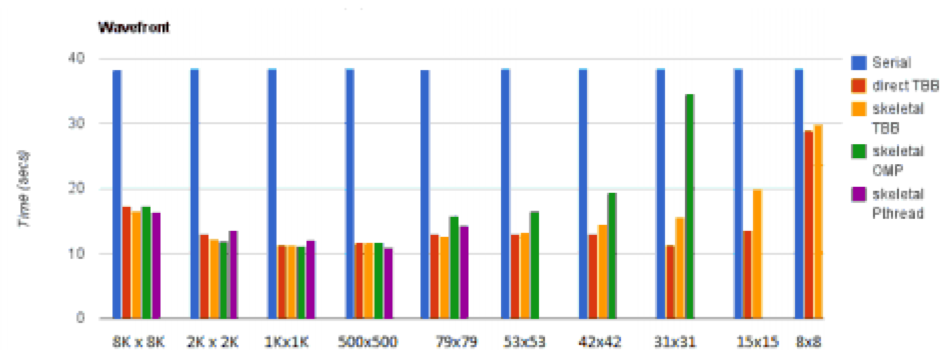


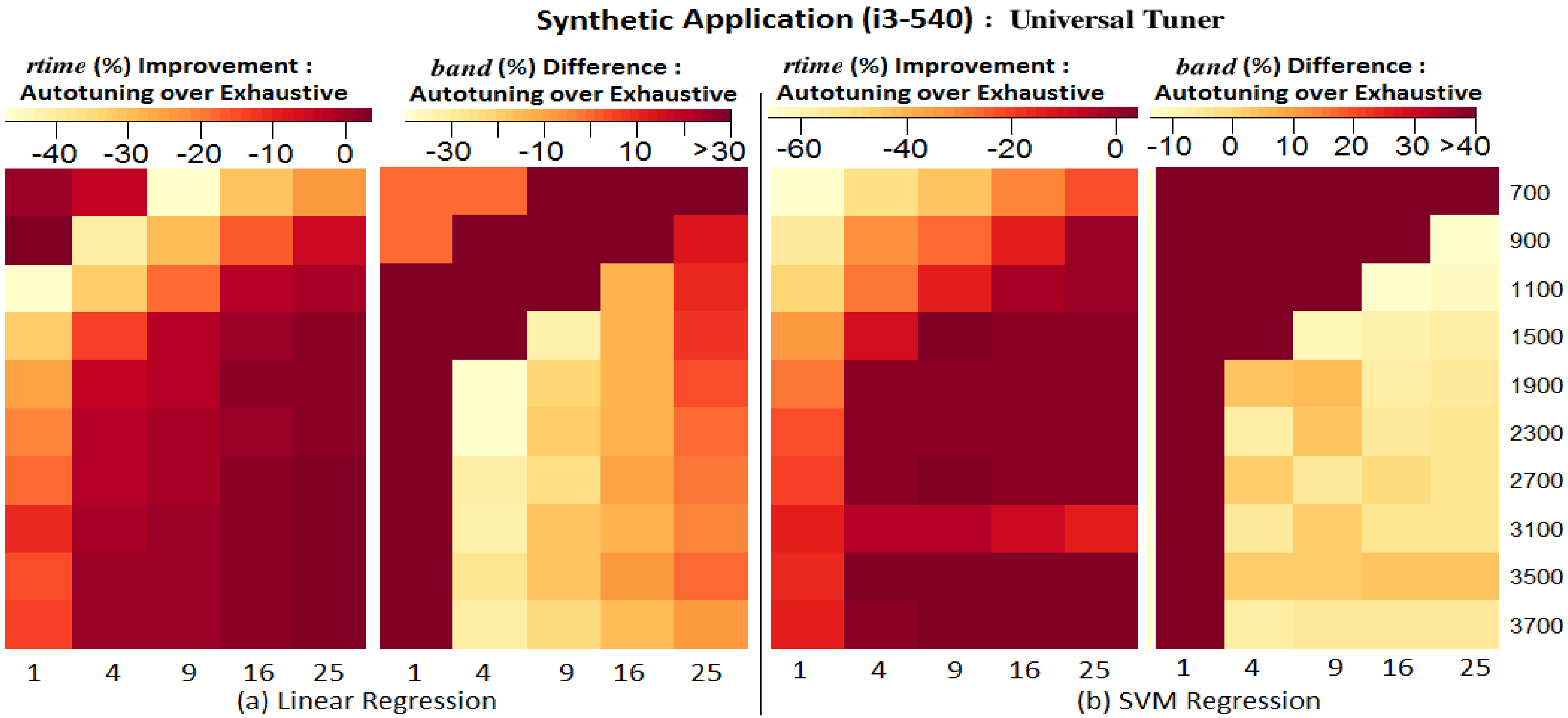
Figure 4.11: Speedup of Nash Kernel





Appendix :Previous Autotuning Performance

- Synthetic Application – note varying colour key



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.

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