ACCELERATING MATRIX LANGUAGES

WITH THE

CELL BROADBAND ENGINE

Raymes Khoury
The University of Sydney

MATLAB and Octave

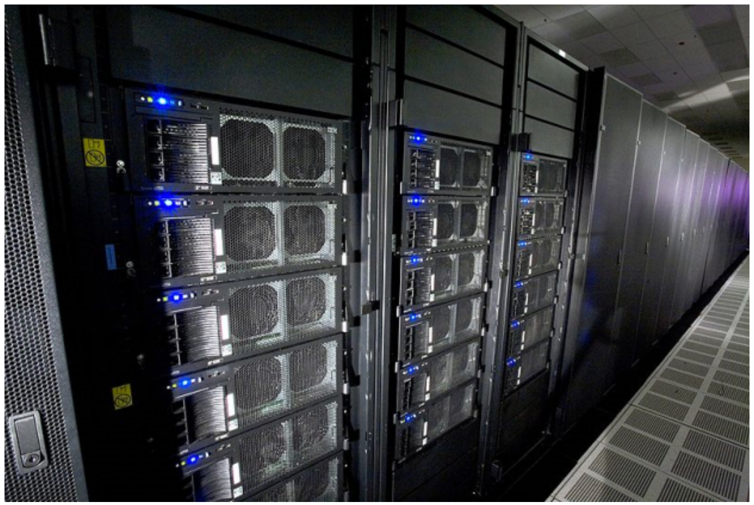
- MATLAB
  - High level, interpreted, un-typed language
  - Very popular among scientists and engineers
  - Simple sequential semantics for expressing algorithms with matrix operations
  - Slow for large problem sizes
- Octave
  - Freely available alternative to MATLAB
  - Part of the GNU project
  - Mimics syntax and semantics of MATLAB
  - Libraries of Octave differ to MATLAB libraries

Modern Parallel Architectures

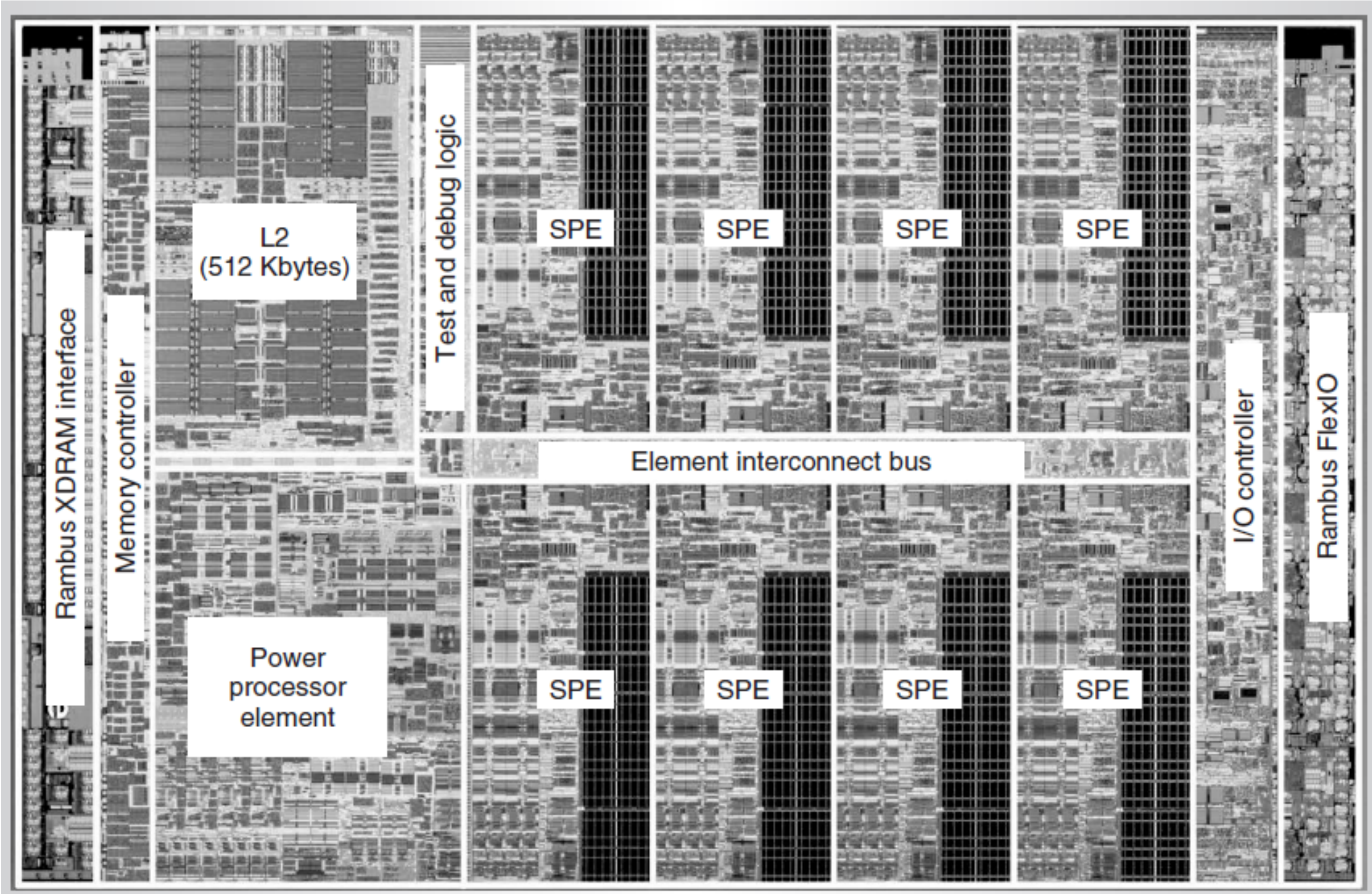
- The limits of performance of traditional single-core processors are reached.
- Fundamental shift towards parallel architectures
- Current popular parallel architectures:
  - Cell Processor (Sony, Toshiba and IBM)
  - Multi-core CPUs (Intel Core2 Series)
  - General Purpose GPUs (Nvidia Tesla)
- Significant boost of performance
  - 15 GFLOPs of a single core vs. 2 TFLOPs

The Cell Broadband Architecture

- Parallel microprocessor architecture
- Developed by Sony, Toshiba and IBM between 2000 and 2005
- Used in the IBM Roadrunner – the worlds fastest supercomputer (Top500, > 1 PETAFLOP)



The Cell Broadband Architecture



Research Questions

- **How do we parallelise a matrix language program for modern parallel architectures?**

Parallelising Matrix Languages

- A) Translate code by hand
  - Concurrent programming is hard
  - Not trained in concurrent programming
  - Expensive/Time consuming
- B) Automatically parallelise code
  - **Our research**

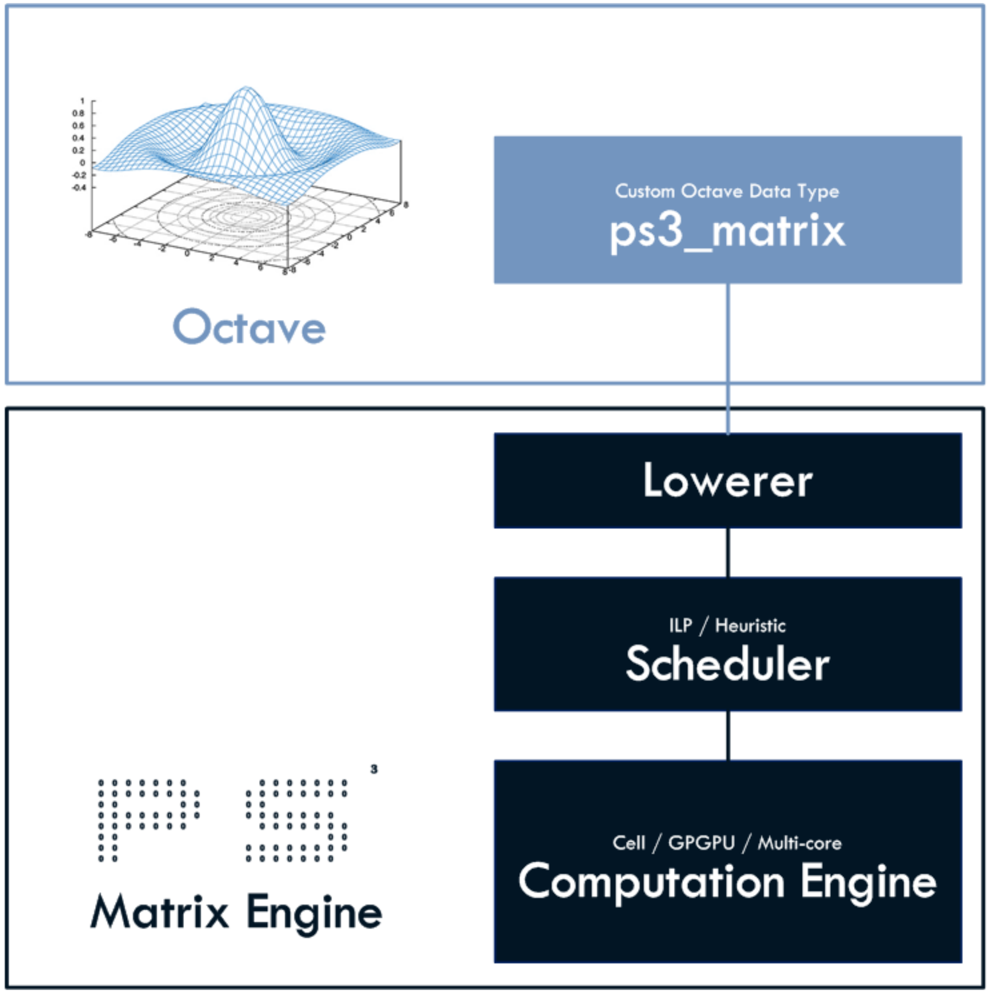
Parallel MATLAB

- 2003 survey found 27 Parallel MATLAB projects
- Limitations
  - Targeted toward distributed parallel architectures
  - Varying degrees of intervention by the programmer required
  - Naive approach
    - Only data parallelism of matrix operations exploited

PS^3: Parallel Octave on the Cell

- Our extension for the Octave interpreter
  - Minimal changes to existing Octave code for programmer
- PS^3 exploits various parallelism in Octave programs:
  - **Data parallelism:** splitting matrices
  - **Instruction level parallelism:** execute independent matrix operations in parallel
  - **Pipeline parallelism**: Communication overlaps with computation
  - **Task parallelism:** concurrent execution of octave programs and matrix operations

Design


Octave
Custom Octave Data Type
ps3_matrix
Lowerer
ILP / Heuristic
Scheduler
Cell / GPGPU / Multi-core
Computation Engine
Matrix Engine

Octave Extension

- Introduced a custom data type called **ps3_matrix**
- To utilise our system convert matrices to **ps3_matrix** matrices

Original code

```
x = rand(100);
y = rand(100);
a = x + y;
b = x .* y;
c = a + b;
disp(c);
```

Parallel code

```
x = ps3_matrix(rand(100));
y = ps3_matrix(rand(100));
a = x + y;
b = x .* y;
c = a + b;
disp(c);
```

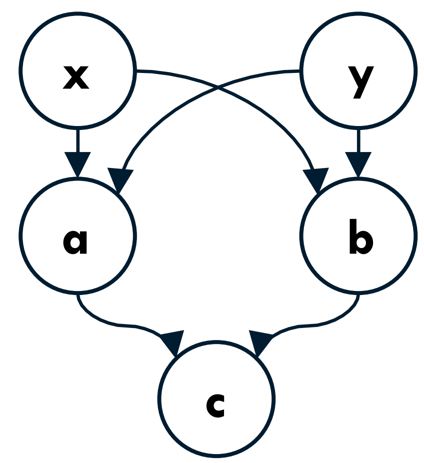
Octave Extension

- Lazy evaluation used to collect traces of operations whose result is not needed
- Data dependence graph of these operations constructed

Source code

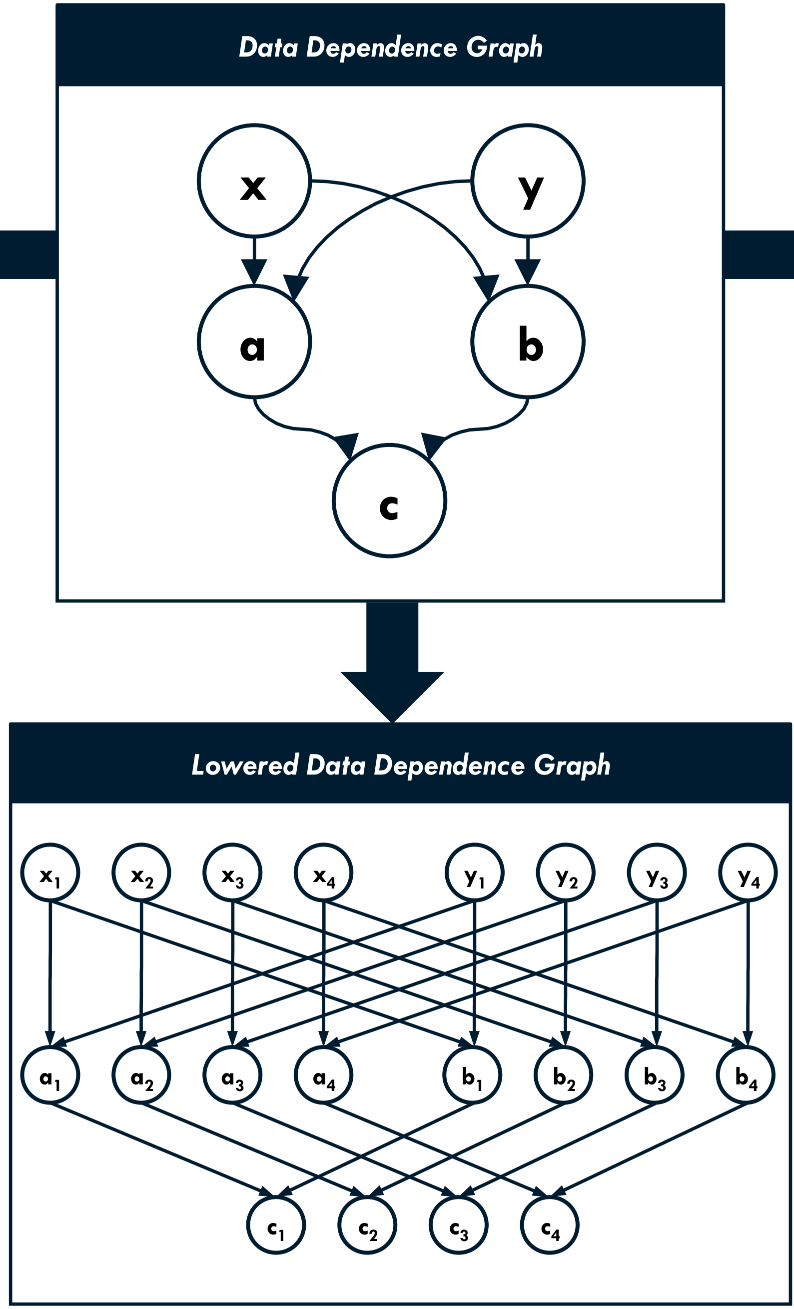
```
x = ps3_matrix(rand(100));
y = ps3_matrix(rand(100));
a = x + y;
b = x .* y;
c = a + b;
disp(c);
```

Data Dependence Graph



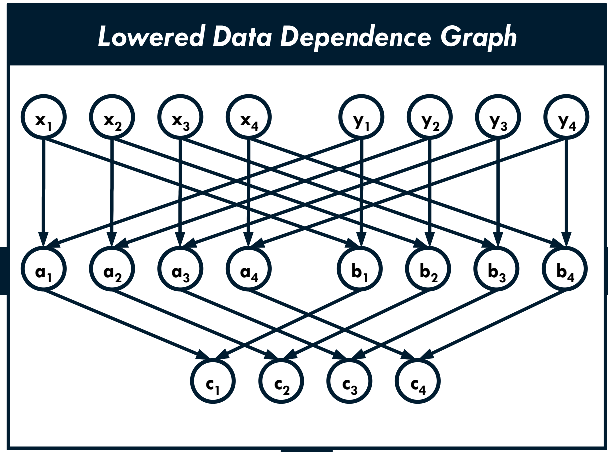
Lowering

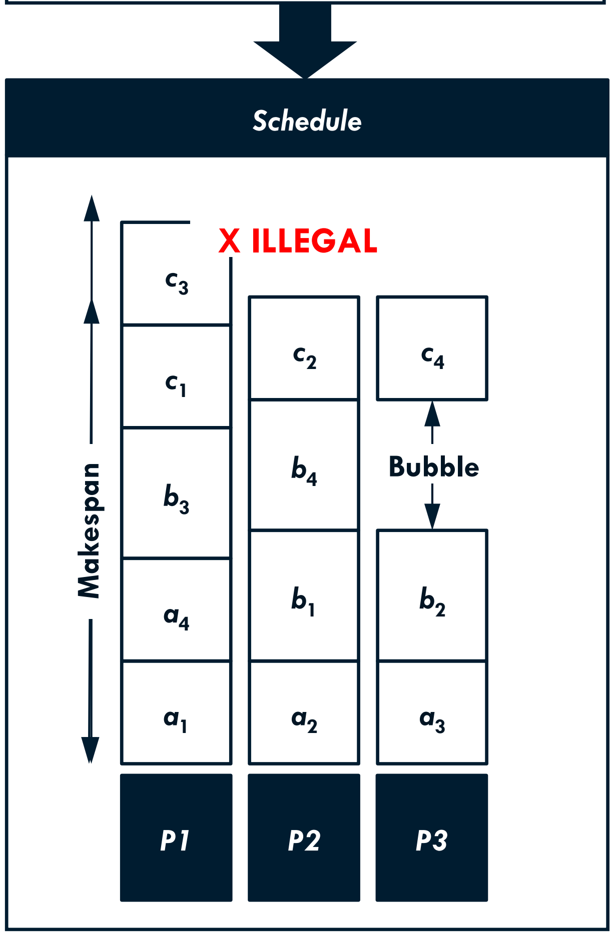
- Lowering is triggered by evaluating a trace, e.g., `disp(c)`
- Matrices are split into sub-matrices
- Parallel computations of sub-matrices on SPUs
- SPEs have 256KB local store
  - Splitting matrices is a necessity!



Scheduler

- Lowered operations are scheduled among the available processors
- Want to schedule in a way that
  - Satisfies data dependencies between operations
  - Minimises the makespan of execution



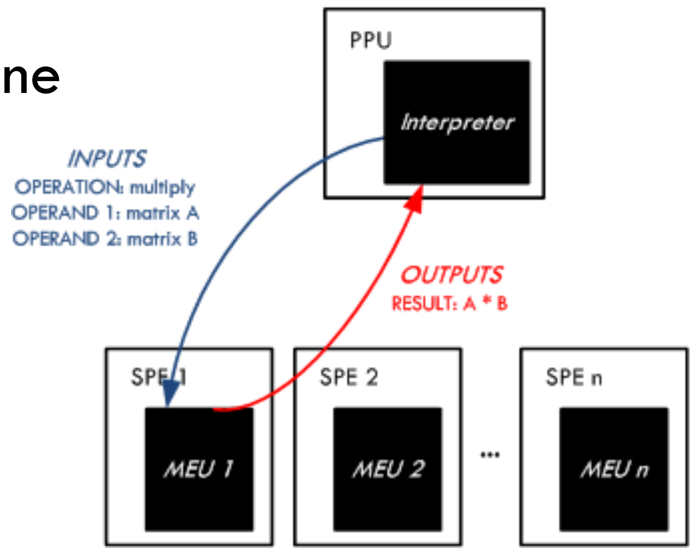


Scheduler

- The scheduling problem is NP-Hard
  - Finding an optimal solution takes too long to do at runtime
- Designed a heuristic
  - Worst-case runtime complexity O(nlogn+m)
  - Earliest instruction is scheduled in first available stream
- Designed an Integer Linear Program formulation
  - Gives optimal solution
  - Validate the precision of the heuristic

Computation Engine

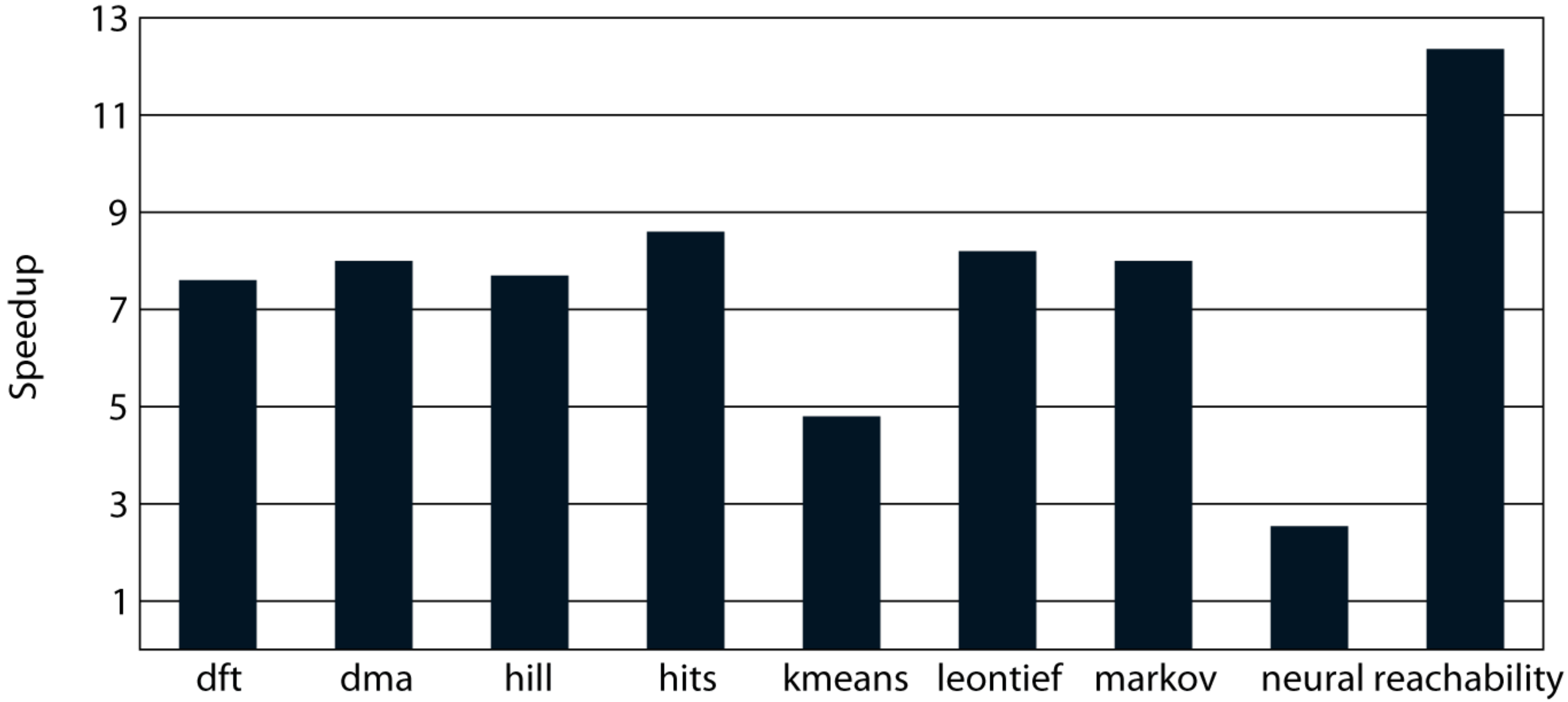
- Computation engine takes schedule and executes it on available processors
- We implement a computation engine for the Cell Processor
- Matrix Execution Units
  - Each SPE runs a small virtual machine for matrix operations
- Features used for performance:
  - Double/Triple buffering
  - SIMD operations



Benchmarks

- 9 benchmark kernels chosen
  - Octave programs that involve many matrix operations
- Include:
  - Computing Markov Chains
  - Computing the Discrete Fourier Transform of a signal
  - K-means clustering
  - Neural network training
- Compared runtime of our system on a Cell processor with Intel Core2Quad processor
  - Q9950, 2.83 GHz

Results: Speedup vs. Intel Core2 Quad



Conclusion

- New system presented for the automatic parallelisation of Octave code on the Cell Processor
  - Exploits several types of parallelism
  - Lazy evaluation to expose instruction level parallelism
  - Schedule operations on processors for maximal utilisation of parallel units
- Results show significant speedups over Octave on more recent and more expensive Intel processors

Related Work

1. Sutter, H.
   The free lunch is over: A fundamental turn toward concurrency in software
   Dr. Dobb's Journal, **2005**, *30*, 202-210
2. Choy, R. & Edelman, A.
   Parallel MATLAB: Doing it Right
   Proceedings of the IEEE, **2005**, *93*, 331-341
3. Fisher, J.
   Trace scheduling: A technique for global microcode compaction
   IEEE Transactions on Computers, **1981**, *100*, 478-490
4. Kwok, Y.-K. & Ahmad, I.
   Static scheduling algorithms for allocating directed task graphs to multiprocessors
   ACM Comput. Surv., ACM, **1999**, *31*, 406-471
5. Chen, T.; Raghavan, R.; Dale, J. N. & Iwata, E.
   Cell broadband engine architecture and its first implementation: a performance view
   IBM J. Res. Dev., IBM Corp., **2007**, *51*, 559-572

Lowered Multiplication

- **b = x * y**

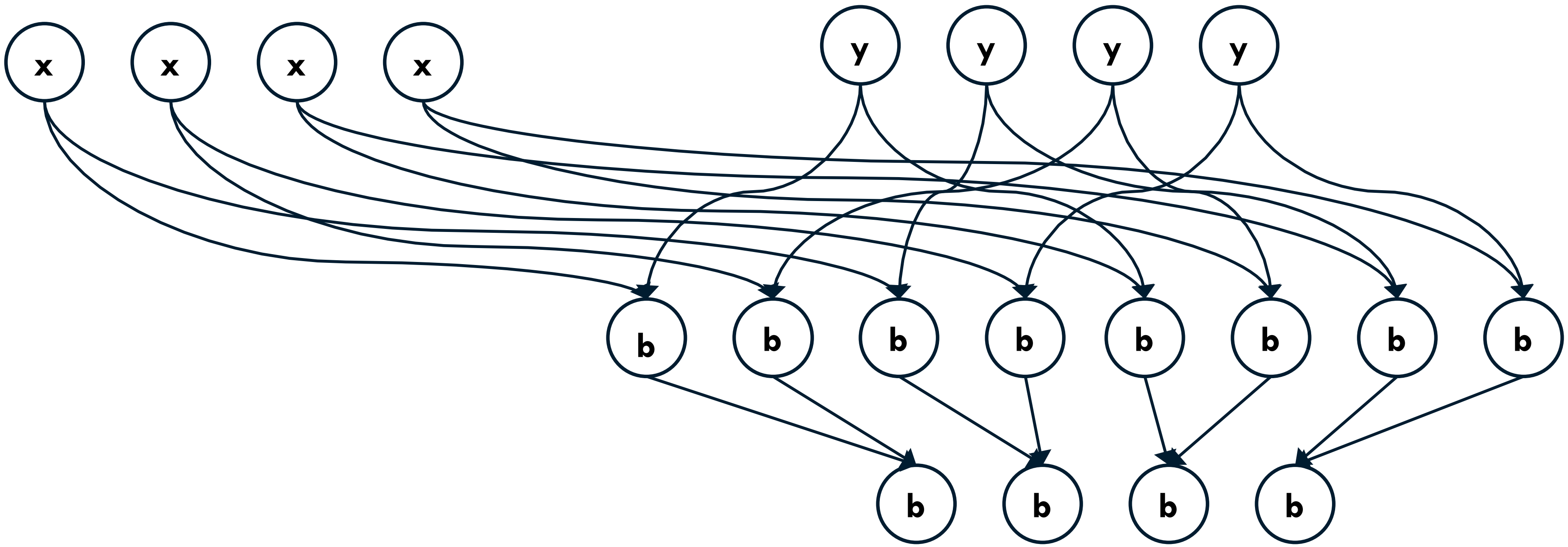
$$\left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right] \times \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right] = \left[\begin{array}{c|c} & \\ \hline & \end{array}\right]$$

$$\left[\begin{array}{c|c} x_{00} & x_{01} \\ \hline x_{10} & x_{11} \end{array}\right] \times \left[\begin{array}{c|c} y_{00} & y_{01} \\ \hline y_{10} & y_{11} \end{array}\right] = \left[\begin{array}{c|c} b_{00} & b_{01} \\ \hline b_{10} & b_{11} \end{array}\right]$$

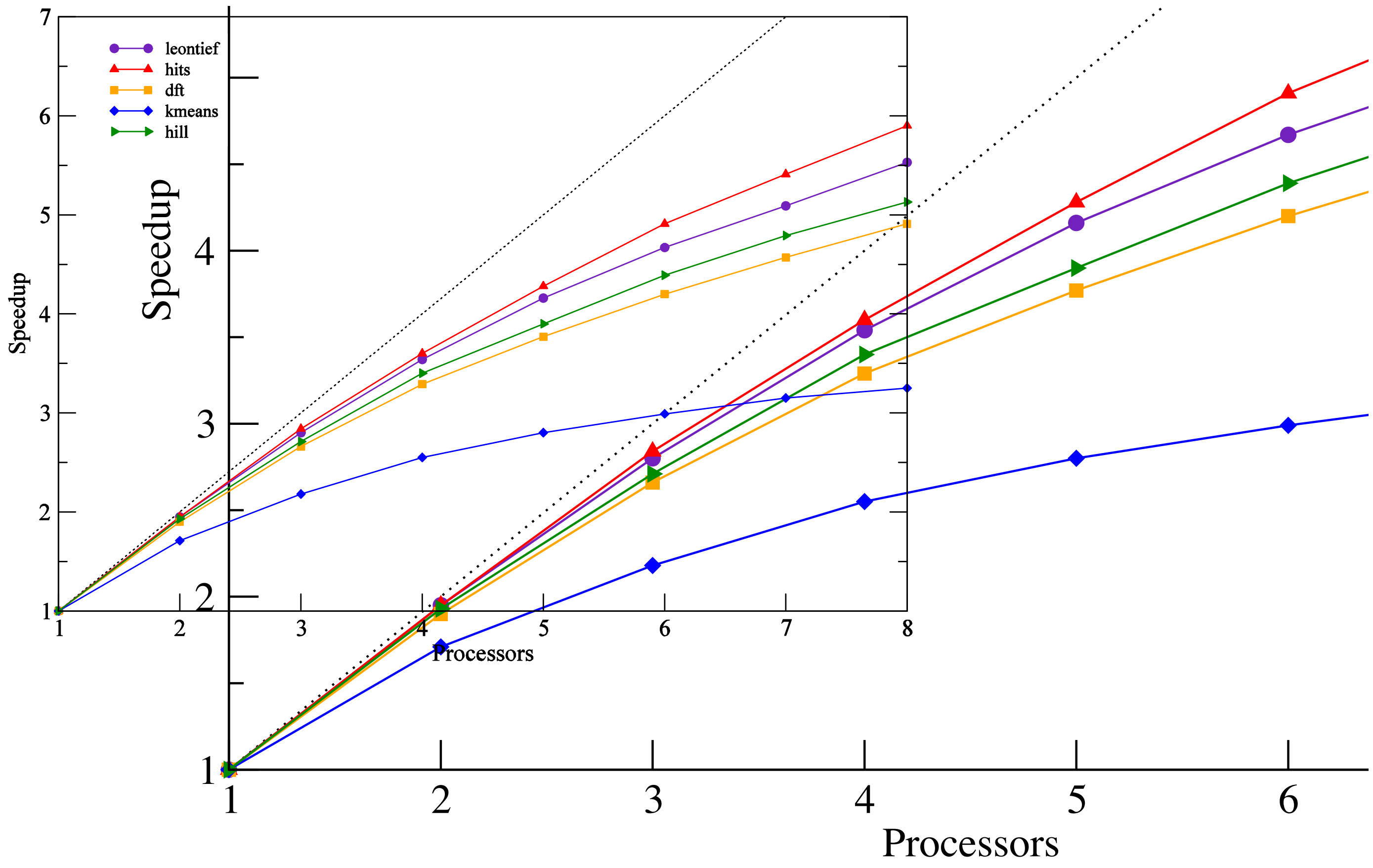
$$b_{00} = x_{00}y_{00} + x_{01}y_{10}$$

Lowered Multiplication

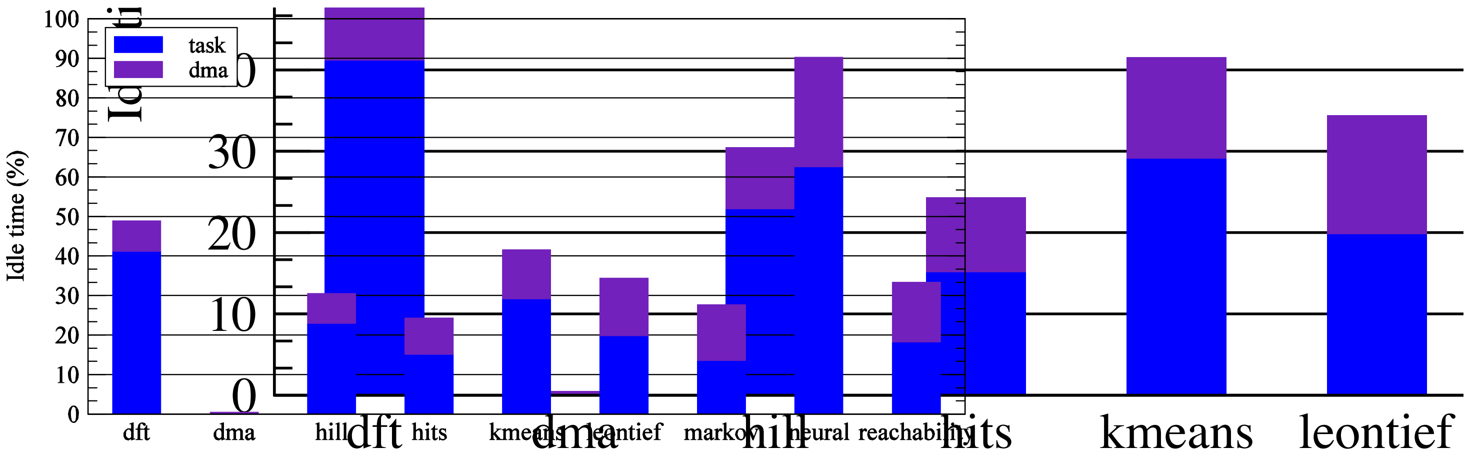
- b = x * y



Efficiency

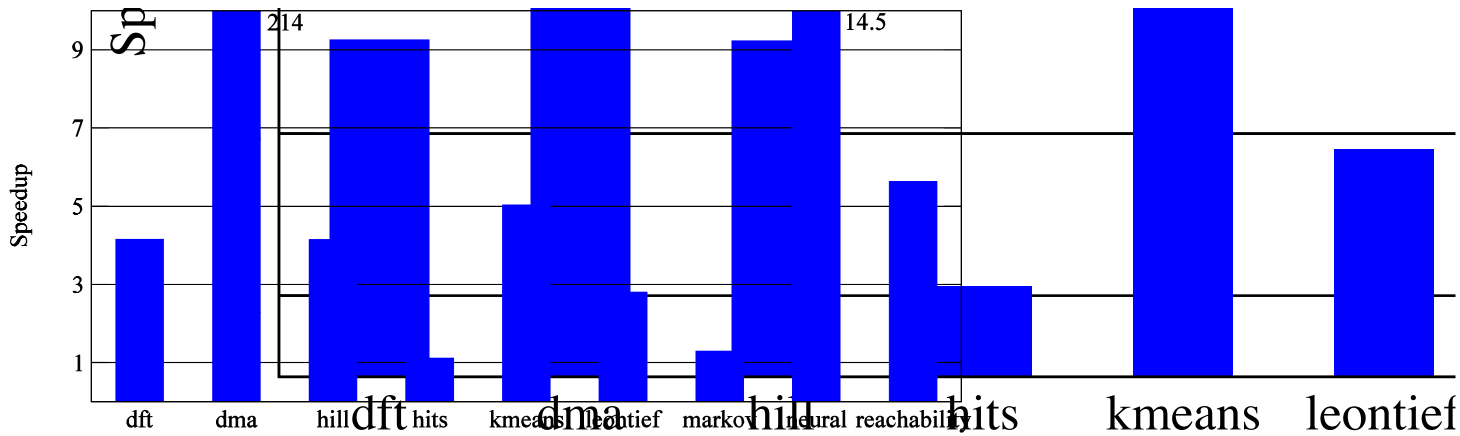

leontief
hits
dft
kmeans
hill
Speedup
Processors
Speedup
Processors

Idle time

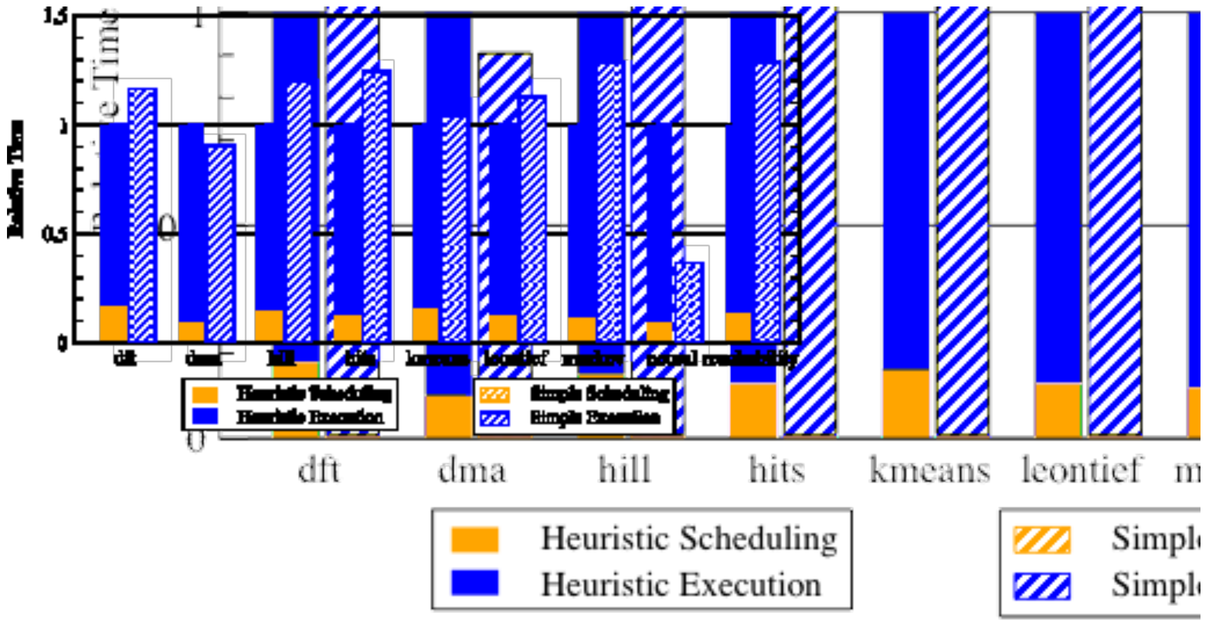


Idle time (%)
100
90
80
70
60
50
40
30
20
10
0
task
dma
dft
dma
hill
hits
kmeans
leontief
markov
neural
reachability

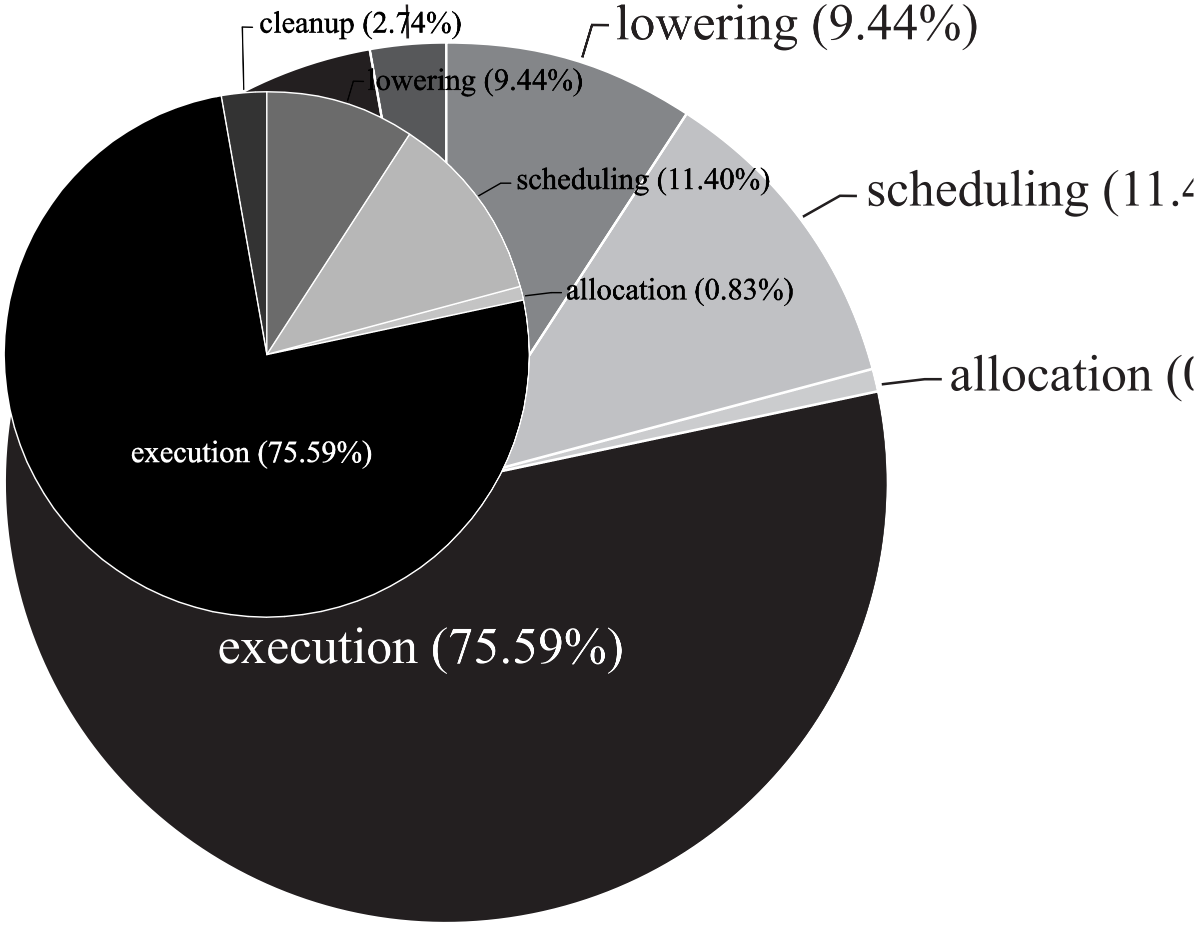
Speedups vs Octave BLAS



Scheduling


dft
dma
hill
hits
kmeans
leontief
Heuristic Scheduling
Heuristic Execution

Breakdown


cleanup (2.74%)
lowering (9.44%)
lowering (9.44%)
scheduling (11.40%)
scheduling (11.4
allocation (0.83%)
allocation (0
execution (75.59%)
execution (75.59%)