

Matrix Multiply Results

Fishy Code Q & A

CS 267 Spring 2005

Yozo Hida

yozo@cs.berkeley.edu

February 25, 2005

Outline

Matrix Multiply Results

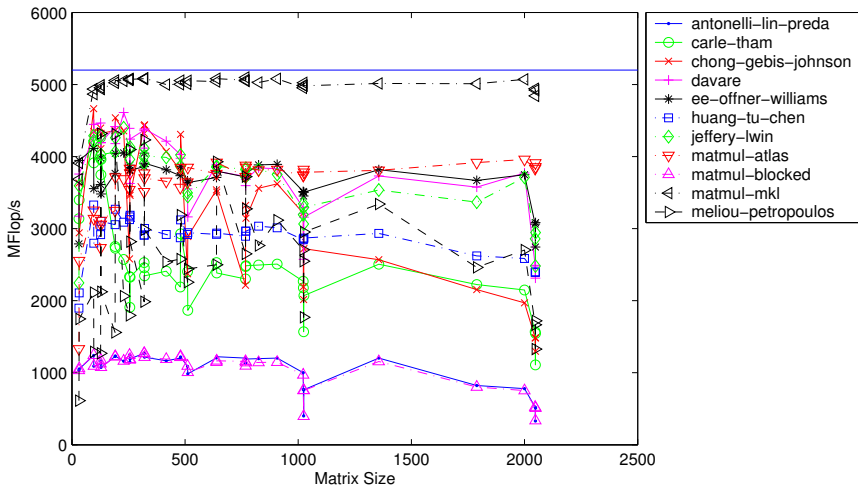
Fishy Code

Outline

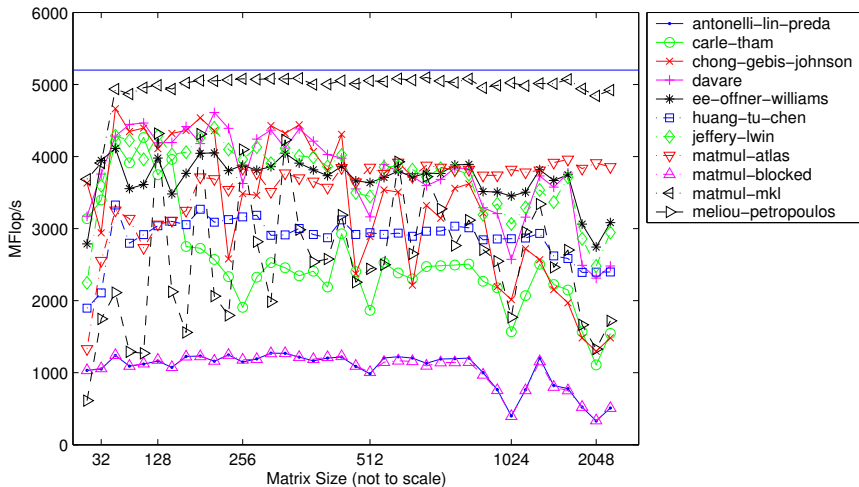
Matrix Multiply Results

Fishy Code

Performance Plot



Performance Plot



Performance Table

Algorithm	max			min			mean		median	
	MF/s	Size	rnk	MF/s	Size	rnk	MF/s	rnk	MF/s	rnk
alp	1271	319	10	330	2048	11	1049	10	1161	10
ct	4265	127	6	1109	2048	8	2546	9	2433	9
cgj	4664	96	2	1298	2048	7	3312	6	3496	6
d	4612	229	3	2314	2048	3	3756	2	3813	3
eow	4113	96	7	2745	2048	2	3684	4	3766	4
htc	3328	96	9	1897	31	5	2872	7	2920	7
jl	4399	229	4	2246	31	4	3707	3	3846	2
mp	4320	128	5	612	31	9	2565	8	2543	8
atlas	3961	2000	8	1331	31	6	3569	5	3752	5
base	1269	320	11	336	2048	10	1039	11	1145	11
mkl	5098	768	1	3686	31	1	4952	1	5024	1

Summary

- ▶ Intel MKL achieves $\approx 98\%$ of peak.
so very little margin of improvement over current best.
Version 7.0 and 7.2 makes a noticeable difference.
- ▶ ATLAS (Automatically Tuned Linear Algebra Software) does did not do as well; some group beat ATLAS results.
- ▶ chong-gebis-johnson and davare has the best performance on smaller matrices (< 512).
- ▶ ee-offner-williams performs best on larger matrices (> 512).
- ▶ jeffery-lwin performs quite well across the board.
- ▶ Top 3 by time-to-completion:

Group	Time
ee-offner-williams	3284.86 s
jeffery-lwin	3390.82 s
davare	3450.14 s

Optimizations Attempted

- ▶ Almost Everyone: register blocking.
Most significant improvements.
- ▶ L2/L3 multilevel blocking.
L2 block size 110-128 most common.
- ▶ Transposing matrix.
Some improvements, but good blocking makes it less useful.
- ▶ Copy to avoid cache conflicts.
- ▶ Loop unrolling. Makes it easier for the compiler to schedule instructions.
- ▶ Software Pipelining (chong-gebis-johnson).
- ▶ Handling of fringe blocks
 - ▶ davare: binary tree method.
 - ▶ huang-tu-chen: padding to nearest multiple of (register) block.

Optimization Not Done

- ▶ Strassen.
 - ▶ requires temporary storage (at least $3M^2$).
 - ▶ less locality in memory reference
 - ⇒ needs more computation to make it efficient.
 - ▶ complication for odd sizes (padding / peeling).
 - ▶ antonelli-lin-preda attempted but could not get it to work.
- ▶ Recursive memory layout

Outline

Matrix Multiply Results

Fishy Code

Fishy Code: General Idea

Fish with constant mass obeys $F = ma$ where F is the force, m is the mass, and a the acceleration. Thus

$$\frac{dv_i}{dt} = \frac{1}{m} \sum_{j \neq i} F_{ij}$$

This is discretized as

$$v_i(t + \Delta t) = v_i(t) + \frac{\Delta t}{m} \sum_{j \neq i} F_{ij}(t)$$

$$x_i(t + \Delta t) = x_i(t) + \Delta t v_i(t)$$

Here we used the slope at time t (which is known), giving the explicit forward Euler.

Fishy Code

Be sure to

- ▶ Profile either OpenMP or pthreads version of code (but no tuning necessary).
- ▶ Profile the original MPI code.
 - ▶ Where are the bottlenecks?
 - ▶ Memory usage?
 - ▶ Scalability?
 - ▶ Communication vs. computation?
 - ▶ Any difference if run on one node or more than one?
- ▶ Do some simple performance modeling on the MPI code.
 - ▶ α - β model (latency vs. bandwidth). Can you estimate machine parameters?
 - ▶ LogP model.
- ▶ Improve the MPI code as outlined in the assignment page.
- ▶ Profile the improved code.

Optimizations

Some optimization not mentioned:

- ▶ Initial fish generation.
- ▶ Exactly which data needs to be sent?
- ▶ How to output data in parallel.

Extras:

- ▶ Play with other kinds of forces. Example:

$$F = \begin{cases} C \left(2 - \frac{1}{60r}\right) & \text{for } r \leq \frac{1}{60} \\ \frac{C}{(60r)^5} & \text{for } r > \frac{1}{60} \end{cases}$$

- ▶ Does spacial decomposition make sense? Why? Try it.

Questions

Any Questions ?