Partitioning Spatially Located Load with Rectangles

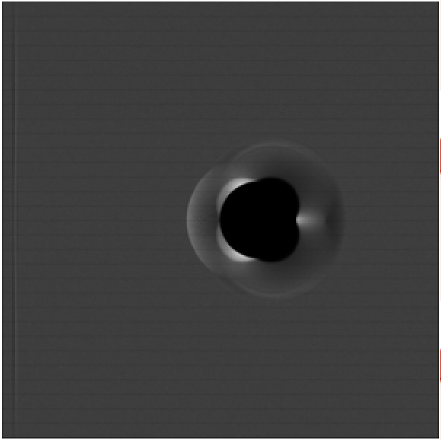
Erik Saule[1], Erdeniz Ö. Baş[1,2], **Ümit V. Çatalyürek**[1,3]

{esaule,erdeniz,umit}@bmi.osu.edu

[1]Department of Biomedical Informatics
[2]Department of Computer Science and Engineering
[3]Department of Electric and Computer Engineering
The Ohio State University

IPDPS 2011

A load distribution problem



Load matrix

In parallel computing, the load can be spatially located. The computation should be distributed accordingly.

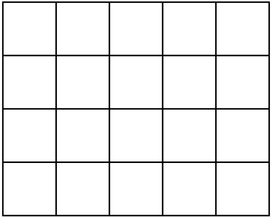
Applications

- Particles in Cell (stencil)
- Sparse Matrices
- Direct Volume Rendering

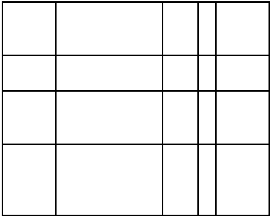
Metrics

- **Load balance**
- Communication
- Stability

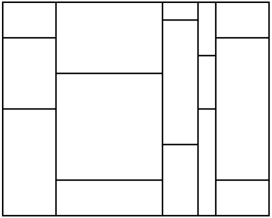
Different kinds of partition



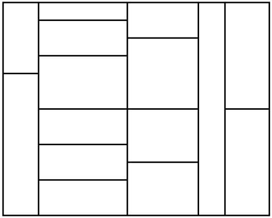
Uniform



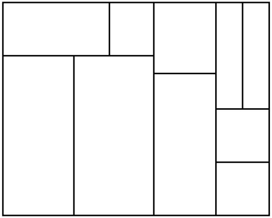
Rectilinear



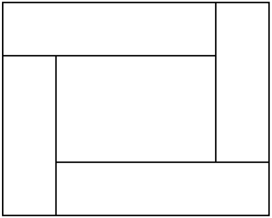
$P \times Q$-way jagged (th)



m-way jagged (def, heur, th, opt)

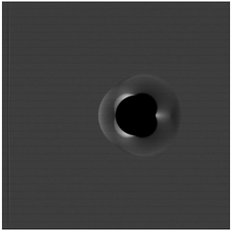


hierarchical (heur, opt)

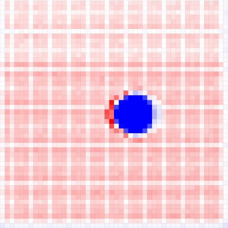


spiral (heur, opt)

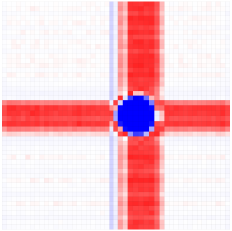
Different load balance on 2304 processors



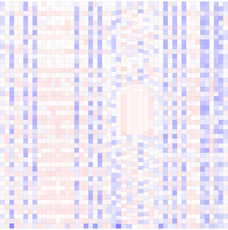
Particles (2050x2050)



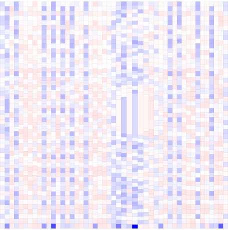
Uniform (17.5%)



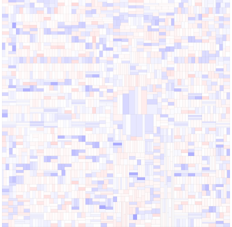
Rectilinear (15.1%)



$P \times Q$-way jagged (2.3%)



m-way jagged (2.0%)



hierarchical (2.7%)

This talk is about how to generate such partitions, either optimally or heuristically, and the type of guarantee we can obtain.

Outline

1. Introduction
2. Preliminaries
   - Notation
   - In One Dimension
   - Simulation Setting
3. Rectilinear Partitioning
   - Nicol's Algorithm
4. Jagged Partitioning
   - $P \times Q$-way Jagged
   - m-way Jagged
5. Hierarchical Bisection
   - Recursive Bisection
   - Dynamic Programming
6. Final thoughts
   - Summing up
   - Conclusion and Perspective

The Rectangular Partitioning Problem

Definition

Let A be a $n_1 \times n_2$ matrix of non-negative values. The problem is to partition the $[1,1] \times [n_1, n_2]$ rectangle into a set S of m rectangles. The load of rectangle $r = [x, y] \times [x', y']$ is $L(r) = \sum_{x \le i \le x', y \le j \le y'} A[i][j]$. The problem is to minimize $L_{max} = \max_{r \in S} L(r)$.

Prefix Sum

Algorithms are rarely interested in the value of a particular element but rather interested in the load of a rectangle. The matrix is given as a 2D prefix sum array Pr such as $Pr[i][j] = \sum_{i' \le i, j' \le j} A[i'][j']$. By convention $Pr[0][j] = Pr[i][0] = 0$.

We can now compute the load of rectangle $r = [x, y] \times [x', y']$ as $L(r) = Pr[x'][y'] - Pr[x-1][y'] - Pr[x'][y-1] + Pr[x-1][y-1]$.

In One Dimension

Optimal : Nicol's algorithm [Nic94] (improved by [PA04])

Based on parametric search.
Complexity: $O((m \log \frac{n}{m})^2)$.

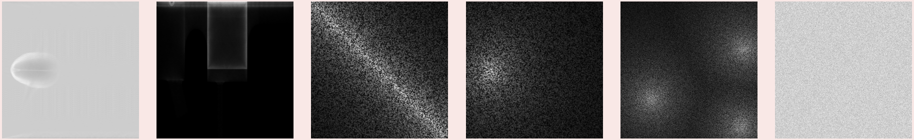
Heuristic : Direct Cut [MP97]

Greedy algorithm.
Complexity: $O(m \log \frac{n}{m})$.
Guarantees : $L_{max}(DC) \leq \frac{\sum_{i'} A[i']}{m} + \max_i A[i]$.

(More details in Section 2.2)

Simulation Setting

Classes (Some inspired by [MS96])



Processors

Simulation are perform with different number of processors: most squared numbers up to 10,000.

Metric

Load imbalance is the presented metric : $\frac{L_{max}}{\frac{\sum_{i,j} A[i][j]}{m}} - 1$.

Outline of the Talk

1 Introduction
2 Preliminaries
- Notation
- In One Dimension
- Simulation Setting

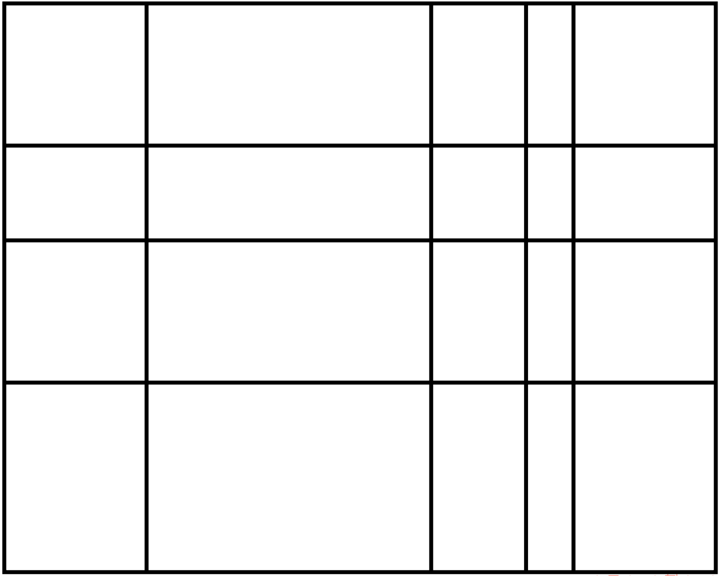
3 Rectilinear Partitioning
- Nicol's Algorithm

4 Jagged Partitioning
- $P \times Q$-way Jagged
- m-way Jagged

5 Hierarchical Bisection
- Recursive Bisection
- Dynamic Programming

6 Final thoughts
- Summing up
- Conclusion and Perspective

Rectilinear Partitioning



Nicol's Algorithm [Nic94]: `RECT-NICOL`

The algorithm

`RECT-NICOL` is an iterative heuristic. At each iteration the partition in one dimension is refined by using a 1D algorithm.
Complexity:

- $O(n_1 n_2)$ iterations (around 10 in practice)
- 1 iteration : $O(Q(P \log \frac{n_1}{P})^2 + P(Q \log \frac{n_2}{Q})^2)$.

Other algorithms

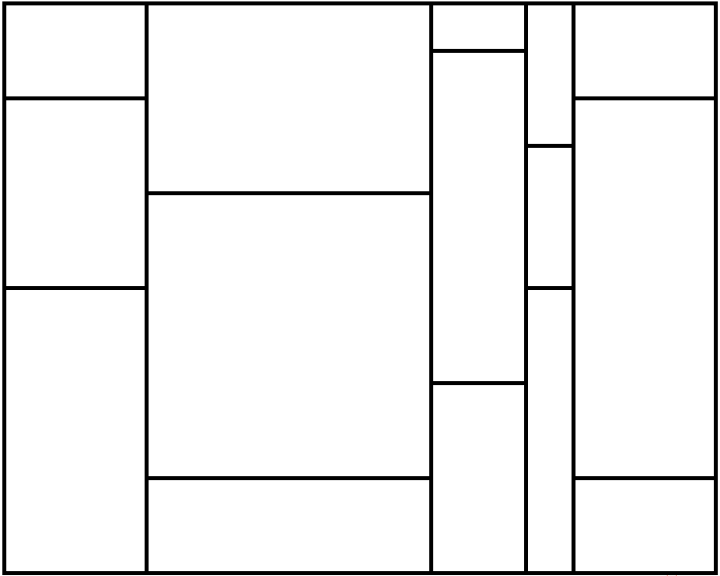
The problem of finding the optimal Rectilinear Partitioning is NP-Complete. Therefore, other algorithms which mainly focuses on theoretical properties. The guarantees are unsuitable. The algorithms are computationally expensive (n_1^{10}) and difficult to implement (rely on linear programming or present numerical instability).

(See Section 3.1 for more details)

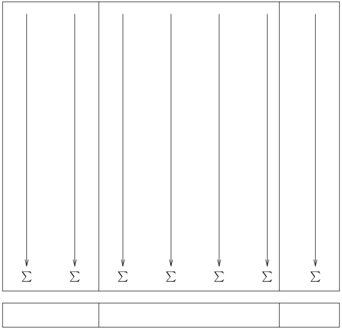
Outline of the Talk

1. Introduction
2. Preliminaries
   - Notation
   - In One Dimension
   - Simulation Setting
3. Rectilinear Partitioning
   - Nicol's Algorithm
4. Jagged Partitioning
   - $P \times Q$-way Jagged
   - m-way Jagged
5. Hierarchical Bisection
   - Recursive Bisection
   - Dynamic Programming
6. Final thoughts
   - Summing up
   - Conclusion and Perspective

$P \times Q$-way Jagged Partitioning



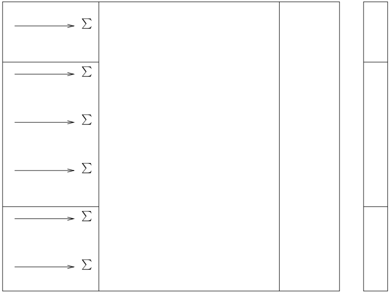
A $P \times Q$-way Jagged Heuristic: JAG-PQ-HEUR



$P \times Q$ Jagged Partitioning

- Sum on columns to generate a 1D problem.
- Partition it in P parts.
- For the first stripe, sum on rows.
- Partition it in Q parts.
- Treat all stripes.

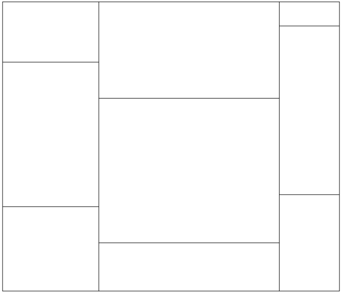
A $P \times Q$-way Jagged Heuristic: JAG-PQ-HEUR



$P \times Q$ Jagged Partitioning

- Sum on columns to generate a 1D problem.
- Partition it in P parts.
- For the first stripe, sum on rows.
- Partition it in Q parts.
- Treat all stripes.

A $P \times Q$-way Jagged Heuristic: JAG-PQ-HEUR



$P \times Q$ Jagged Partitioning

- Sum on columns to generate a 1D problem.
- Partition it in P parts.
- For the first stripe, sum on rows.
- Partition it in Q parts.
- Treat all stripes.

Complexity :
$O((P \log \frac{n_1}{P})^2 + P \times (Q \log \frac{n_2}{Q})^2)$.

How good is that ?

Theorem (Theorem 1 in Section 3.2.1)

If there are no zero in the array, JAG-PQ-HEUR is a $(1+\Delta\frac{P}{n_1})(1+\Delta\frac{Q}{n_2})$-approximation algorithm where $\Delta = \frac{\max A}{\min A}$, $P < n_1$, $Q < n_2$.

Proof.

Based on the guarantee of 1D heuristics. □

Theorem (Theorem 2 in Section 3.2.1)

The approximation ratio is minimized by $P = \sqrt{m\frac{n_1}{n_2}}$.

An optimal $P \times Q$-way jagged partitioning : JAG-PQ-OPT

A Dynamic Programming Formulation

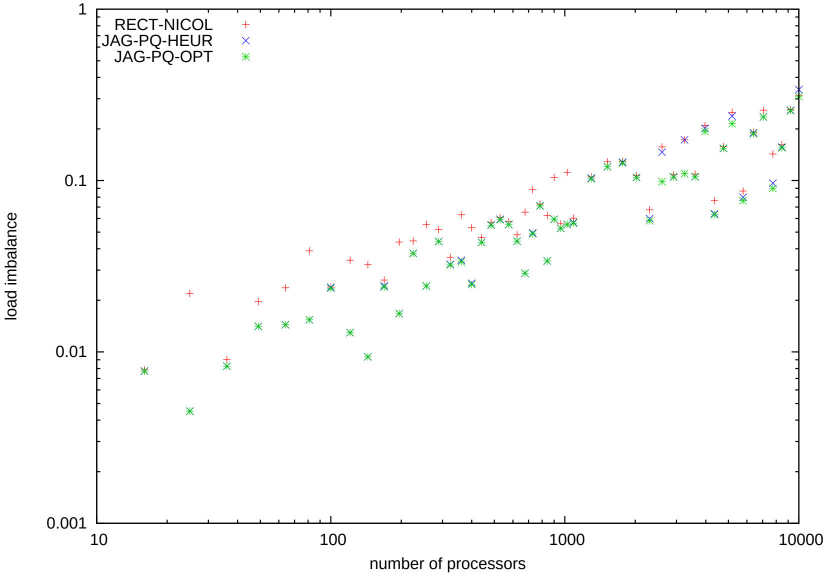
$$\begin{cases} L_{max}(n_1, P) = \min_{1 \le k < n_1} \max(L_{max}(k-1, P-1), 1D(k, n_1, Q)) \\ L_{max}(0, P) = 0 \\ L_{max}(n_1, 0) = +\infty, \forall n_1 \ge 1 \end{cases}$$

- $O(n_1 P)$ L_{max} functions to evaluate. (Each is $O(k)$.)
- $O(n_1^2)$ 1D functions to evaluate. (Each is $O((Q \log \frac{n_2}{Q})^2)$.)

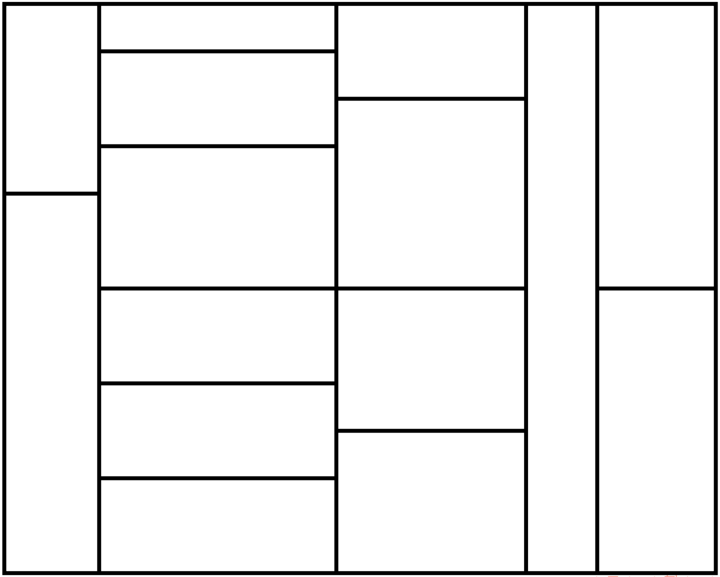
(Some significant implementation optimizations apply)

For a 512x512 matrix and 1000 processors, that's 512,000+262,144 values. On 64-bit values, that's 6MB.

Performance of $P \times Q$-way jagged (PIC-MAG it=30000)



m-way Jagged Partitioning



m-way jagged partitioning heuristic: JAG-M-HEUR

Algorithm

Cut in P stripes. Distribute processors in each stripe proportionally to the stripe's load : $alloc_j = \left\lceil \frac{\sum_{i,j} A[i][j]}{load_j}(m - P) \right\rceil$.

m-way jagged partitioning heuristic: JAG-M-HEUR

Algorithm

Cut in P stripes. Distribute processors in each stripe proportionally to the stripe's load : $alloc_j = \left\lceil \frac{\sum_{i,j} A[i][j]}{load_j}(m-P) \right\rceil$.

Theorem (Theorem 3 in Section 3.2.2)

If there are no zero in A, the approximation ratio of the described algorithm is $\frac{\mathbf{m}}{\mathbf{m-P}}(1+\boldsymbol{\Delta}\frac{\mathbf{1}}{\mathbf{n_2}}) + \frac{m\Delta}{Pn_2}(1+\frac{\Delta P}{n_1})$.

Proof.

Same kind of proof than for heuristic $P \times Q$ jagged partitioning. □

Recall that the guarantee of heuristic $P \times Q$ jagged partitioning was: $(1+\boldsymbol{\Delta}\frac{\mathbf{P}}{\mathbf{n_1}}) + \frac{m\Delta}{Pn_2}(1+\frac{\Delta P}{n_1})$. m-way is better for large m values.

An optimal m-way partitioning JAG-M-OPT

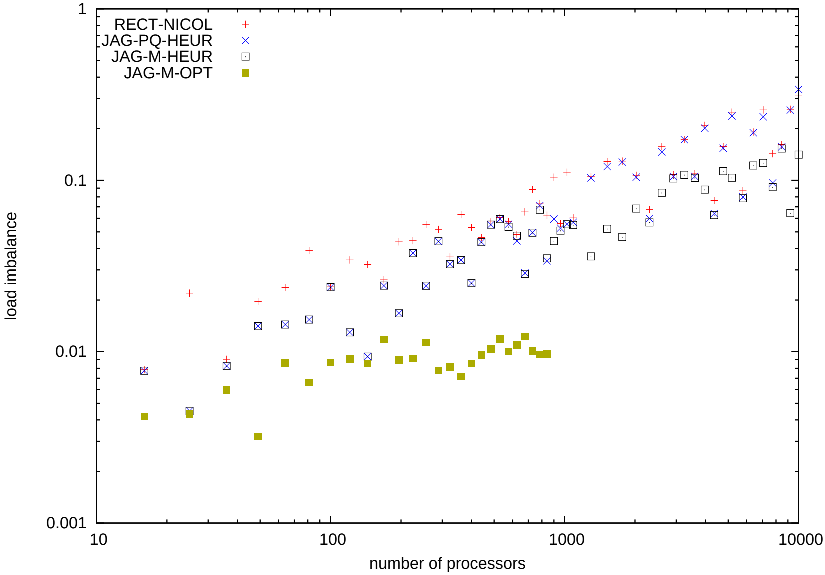
A Dynamic Programming Formulation

$$\begin{cases} L_{max}(n_1, m) = \min_{1 \le k < n_1, 1 \le x \le m} \max(L_{max}(k-1, m-x), 1D(k, n_1, x)) \\ L_{max}(0, m) = 0 \\ L_{max}(n_1, 0) = +\infty, \forall n_1 \ge 1 \end{cases}$$

- $O(n_1 m)$ L_{max} functions.
- $O(n_1^2 m)$ 1D functions. (m times more than for $P \times Q$ jagged)

(The same kind of optimizations apply.)
For a 512x512 matrix on 1,000 processors. That's 512,000 + 262,144,000 values, if they are 64-bits, about 2GB (and takes 30 minutes).

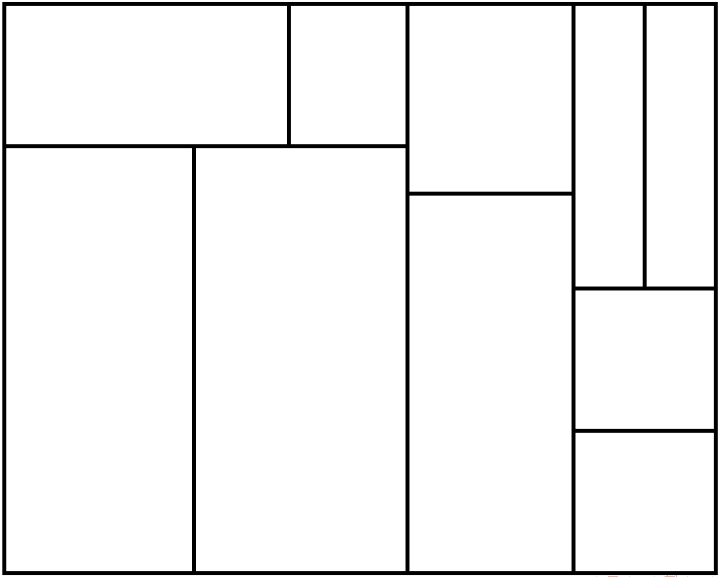
Performance of *m*-way jagged (PIC-MAG it=30000)



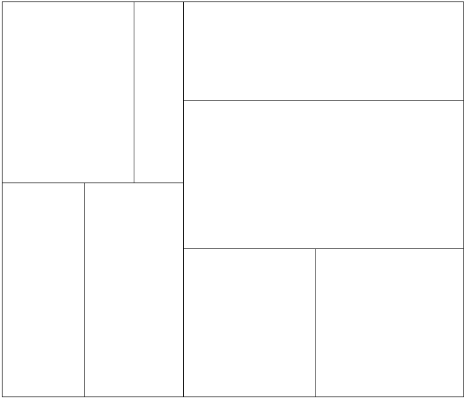
Outline of the Talk

1. Introduction
2. Preliminaries
   - Notation
   - In One Dimension
   - Simulation Setting
3. Rectilinear Partitioning
   - Nicol's Algorithm
4. Jagged Partitioning
   - $P \times Q$-way Jagged
   - m-way Jagged
5. Hierarchical Bisection
   - Recursive Bisection
   - Dynamic Programming
6. Final thoughts
   - Summing up
   - Conclusion and Perspective

Hierarchical Bisection Partitioning



Recursive Bisection [BB87]: `HIER-RB`

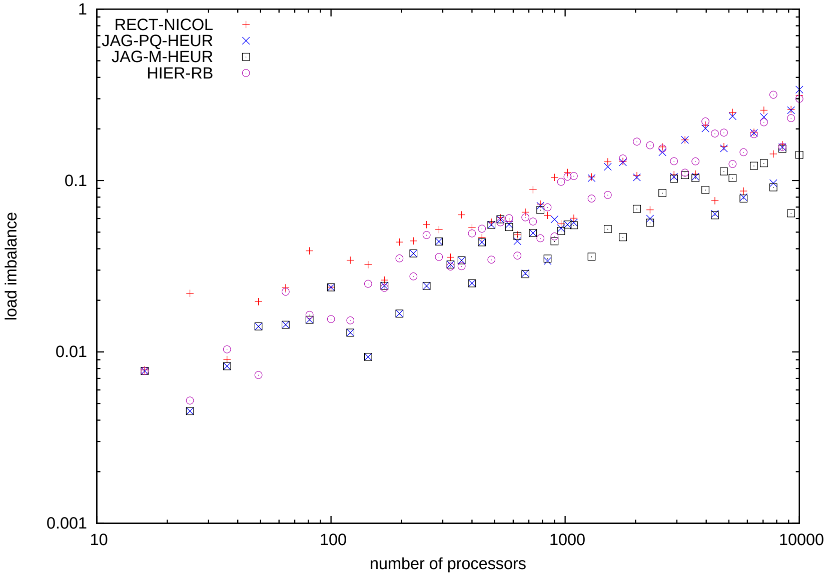


Algorithm

- m processors to partition a rectangle.
- Cut to balance the load evenly.
- Allocate half the processors to each side.
- Cut the dimension that balances the load best.

Complexity: $O(m \log \max n_1, n_2)$.

Performance of HIER-RB (PIC-MAG it=30000)



An Optimal Hierarchical Bisection Algorithm

A Dynamic Programming Formulation

$$L_{max}(x_1, x_2, y_1, y_2, m) = \min_j \min(\\ \min_x \max(L_{max}(x_1, x, y_1, y_2, j), L_{max}(x+1, x_2, y_1, y_2, m-j))\\ , \min_y \max(L_{max}(x_1, x_2, y_1, y, j), L_{max}(x_1, x_2, y+1, y_2, m-j)))$$

- $O(n_1^2 n_2^2 m)$ L_{max} functions. (n_2^2 times more than m-way jagged)

For a 512x512 matrix and 1000 processors, that's 68,719,476,736,000 values. On 64-bit values, that's 544TB.

An Optimal Hierarchical Bisection Algorithm

A Dynamic Programming Formulation

$$
\begin{aligned}
L_{max}(x_1, x_2, y_1, y_2, m) = \min_j \min(\\
&\min_x \max(L_{max}(x_1, x, y_1, y_2, j), L_{max}(x+1, x_2, y_1, y_2, m-j)) \\
&, \min_y \max(L_{max}(x_1, x_2, y_1, y, j), L_{max}(x_1, x_2, y+1, y_2, m-j)))
\end{aligned}
$$

- $O(n_1^2 n_2^2 m)$ L_{max} functions. (n_2^2 times more than m-way jagged)

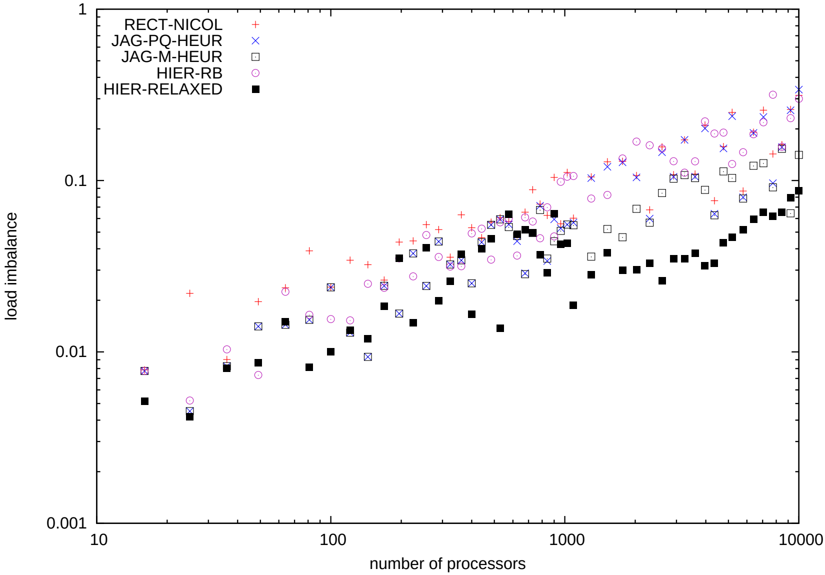
For a 512x512 matrix and 1000 processors, that's 68,719,476,736,000 values. On 64-bit values, that's 544TB.

The Relaxed Hierarchical Heuristic: `HIER-RELAXED`

Build the solution according to

$$
\begin{aligned}
L_{max}(x_1, x_2, y_1, y_2, m) = \min_j \min(\\
&\min_x \max\left(\frac{L(x_1, x, y_1, y_2)}{j}, \frac{L(x+1, x_2, y_1, y_2)}{m-j}\right) \\
&, \min_y \max\left(\frac{L(x_1, x_2, y_1, y)}{j}, \frac{L(x_1, x_2, y+1, y_2)}{m-j}\right))
\end{aligned}
$$

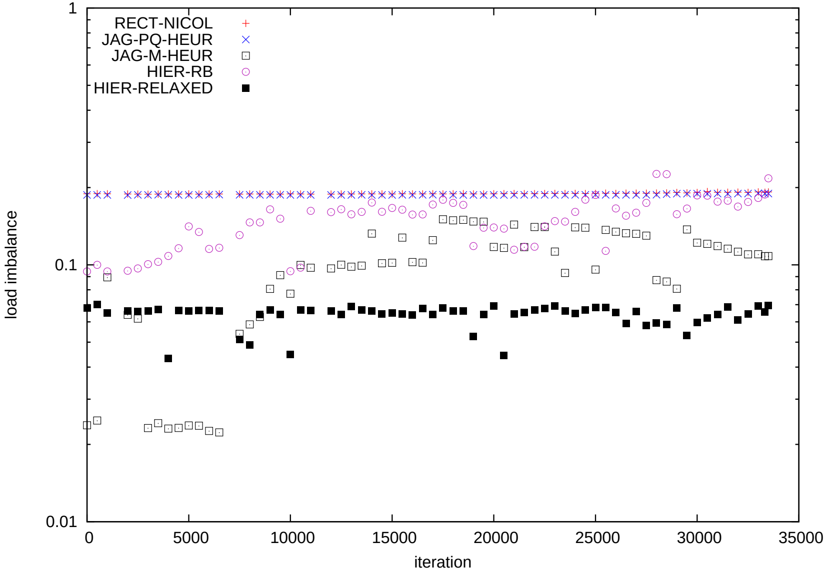
Performance of HIER-RELAXED (PIC-MAG it=30000)



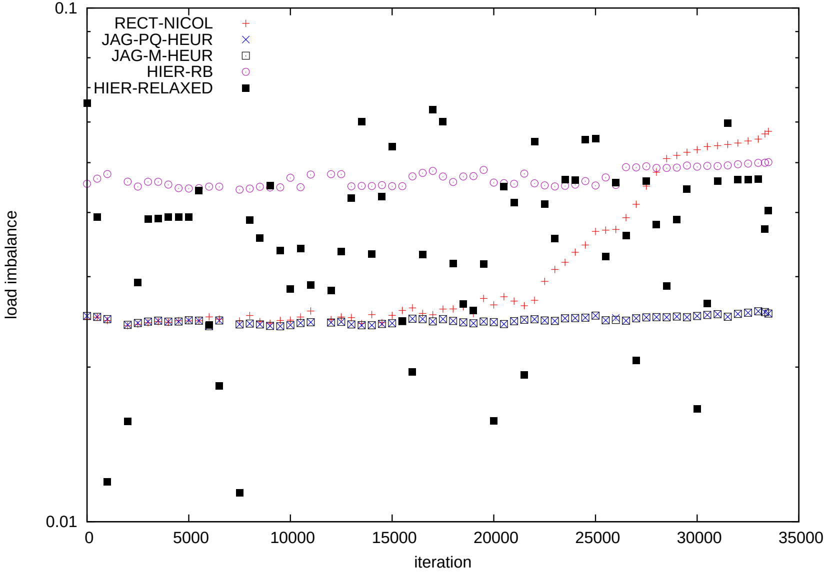
Outline of the Talk

1. Introduction
2. Preliminaries
   - Notation
   - In One Dimension
   - Simulation Setting
3. Rectilinear Partitioning
   - Nicol's Algorithm
4. Jagged Partitioning
   - $P \times Q$-way Jagged
   - m-way Jagged
5. Hierarchical Bisection
   - Recursive Bisection
   - Dynamic Programming
6. Final thoughts
   - Summing up
   - Conclusion and Perspective

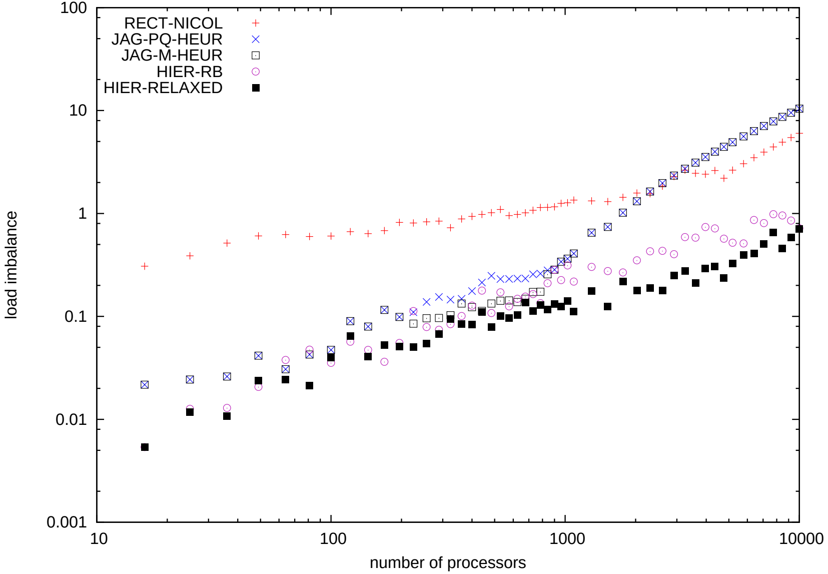
Performance Over the Execution of PIC-MAG (m =6400)



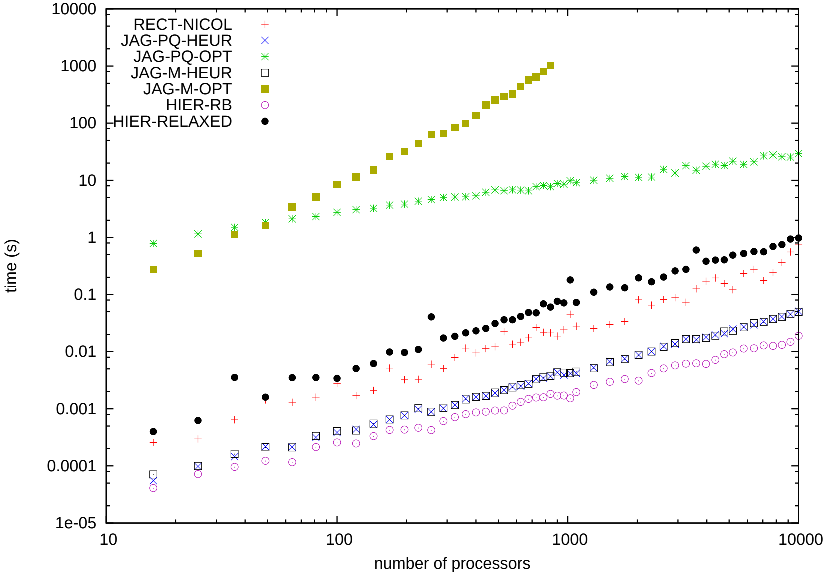
Relaxed Hierarchical Might Be Unstable (m =400)



Sparsity (SLAC)



Runtime on PIC-MAG (it=30000)



What should I use?

Quality

- JAG-M-HEUR and HIER-RELAXED dominates. (Best of two?)
- HIER-RELAXED is better in sparse cases (Figure 14).
- JAG-M-HEUR ties with HIER-RELAXED on dense cases (Figure 12/13).
- But HIER-RELAXED is unstable: it gives very different solutions when run on similar instances (Figure 11).

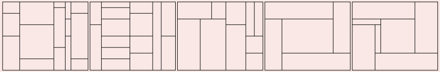
Runtime on a 514x514 matrix with 1024 processors (Figure 6)

- HIER-RB, JAG-PQ-HEUR, JAG-M-HEUR: a few milliseconds.
- HIER-RELAXED, RECT-NICOL: half a second.
- JAG-PQ-OPT: a few seconds.
- JAG-M-OPT: hours.

Conclusion and Perspective

Conclusion

- Proposed a class of partitioning (*m*-way jagged).
- Proved that most recursively defined classes are polynomial:



.

- Proposed two new well-founded heuristics, `JAG-M-HEUR` and `HIER-RELAXED`, which outperform state-of-the-art algorithms.
- Theoretically analyzed `JAG-M-HEUR` and `JAG-PQ-HEUR`.

Perspective

- Better *m*-way jagged partitioning algorithm. (see arXiv 1104.2566)
- Include communication models.
- Integration into a real application. (do you have one ?)

Thank you

Datasets

Thanks to Y. Omelchenko and H. Karimabadi for providing PIC-MAG data; and R. Lee, M. Shephard, and X. Luo for the SLAC data.

More information

contact : umit@bmi.osu.edu

visit: `http://bmi.osu.edu/hpc/` or `http://bmi.osu.edu/~umit`

Research at HPC lab is funded by

Marsha Berger and Shahid Bokhari.
A partitioning strategy for nonuniform problems on multiprocessors.
IEEE Transaction on Computers, C36(5):570–580, 1987.

Serge Miguet and Jean-Marc Pierson.
Heuristics for 1d rectilinear partitioning as a low cost and high quality answer to dynamic load balancing.
In *HPCN Europe '97: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, pages 550–564, London, UK, 1997. Springer-Verlag.

Fredrik Manne and Tor Sørevik.
Partitioning an array onto a mesh of processors.
In *PARA '96: Proceedings of the Third International Workshop on Applied Parallel Computing, Industrial Computation and Optimization*, pages 467–477, London, UK, 1996. Springer-Verlag.

David Nicol.
Rectilinear partitioning of irregular data parallel computations.
Journal of Parallel and Distributed Computing, 23:119–134, 1994.

Ali Pinar and Cevdet Aykanat.
Fast optimal load balancing algorithms for 1d partitioning.
Journal of Parallel and Distributed Computing, 64:974–996, 2004.