Load-Balancing Spatially Located Computations using Rectangular Partitions

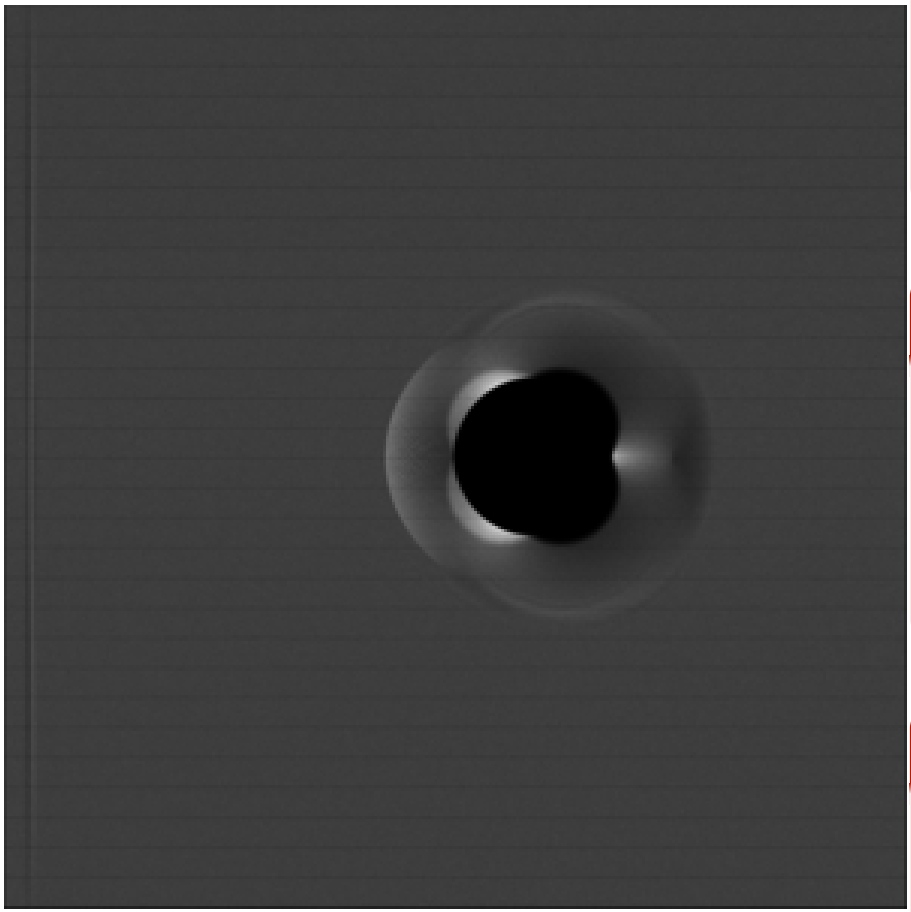
Erdeniz Ö. Baş[1,2], Erik Saule[1], **Ümit V. Çatalyürek**[1,3]

{erdeniz,esaule,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

SIAM Conference on Parallel Processing for Scientific Computing 2012

A load distribution problem



Load matrix

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

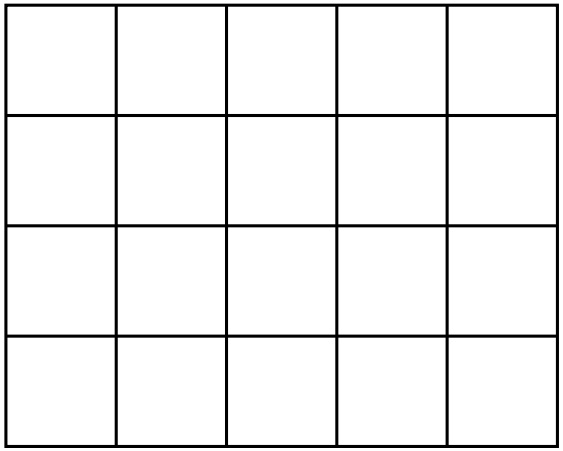
Applications

- Particles in Cell
- Sparse Matrices
- Direct Volume Rendering

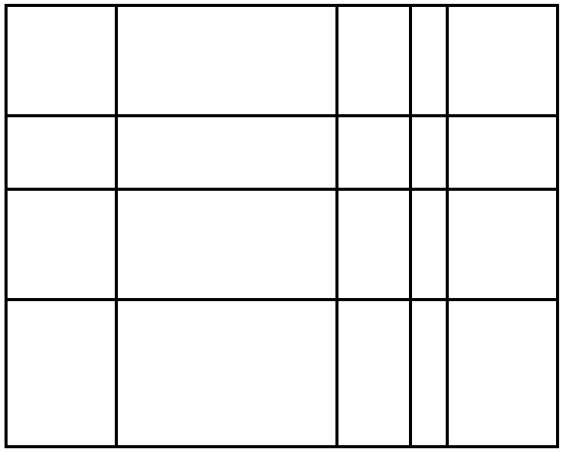
Metrics

- **Load balance**
- Communication
- Stability

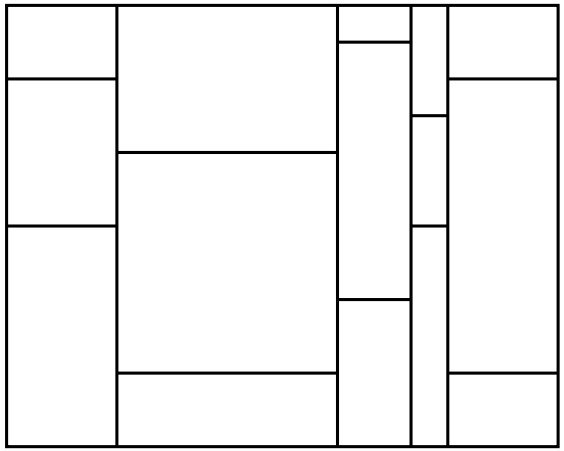
Different kinds of partition



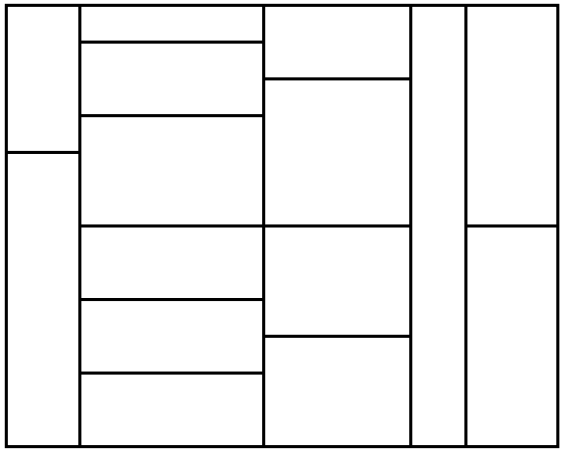
Uniform



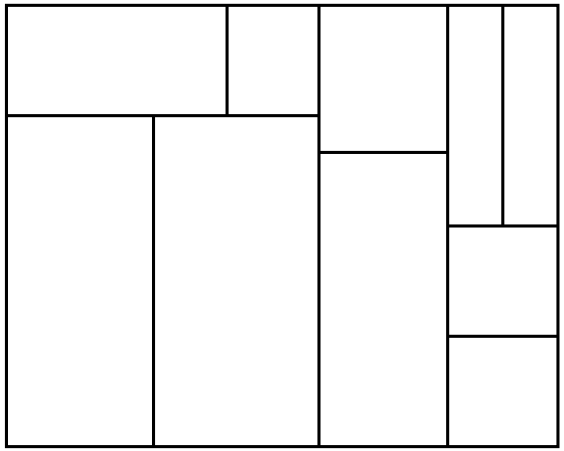
Rectilinear



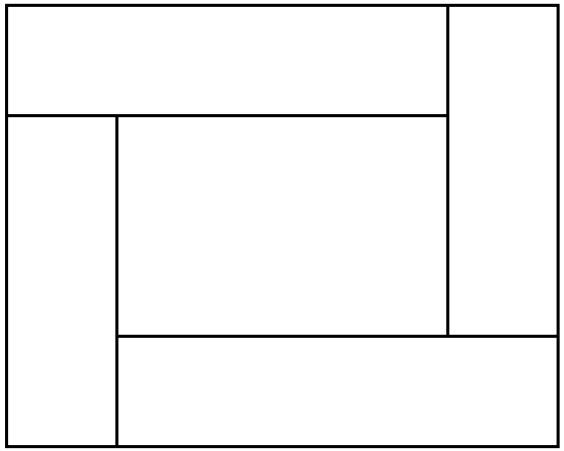
$P \times Q$-way jagged (th)



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

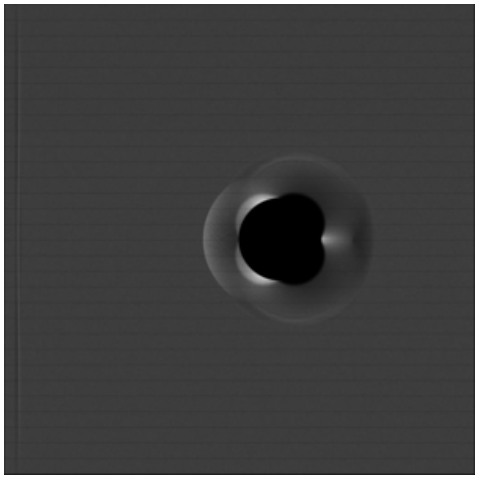


hierarchical (heur, opt)

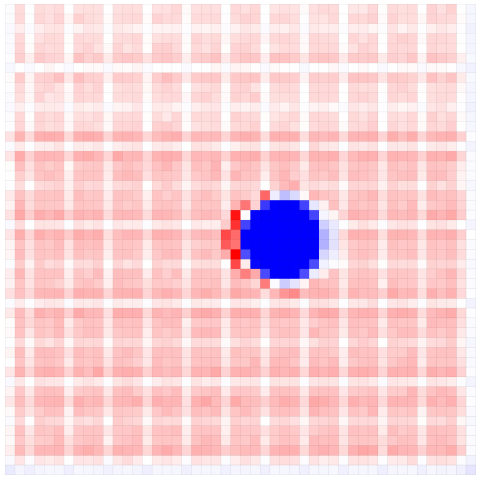


spiral (heur, opt)

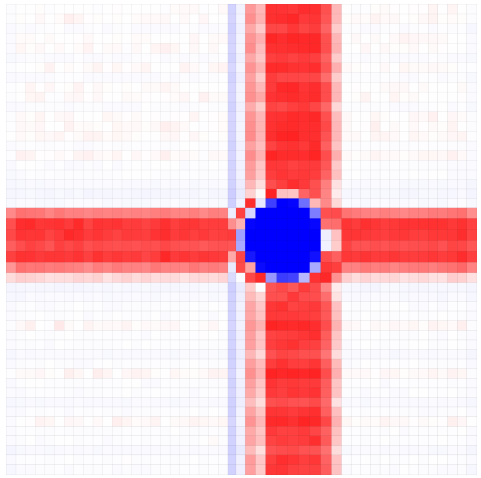
Different load balance on 2304 processors



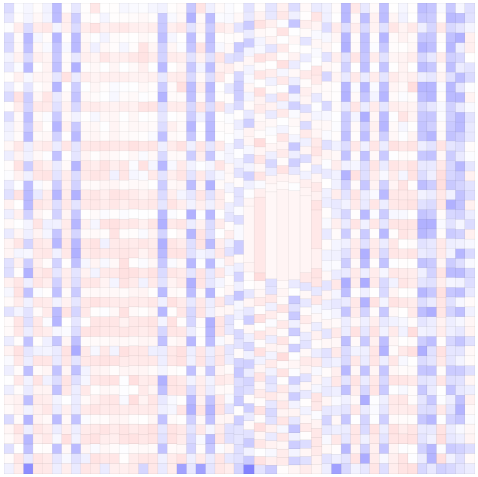
Particles (2050x2050)



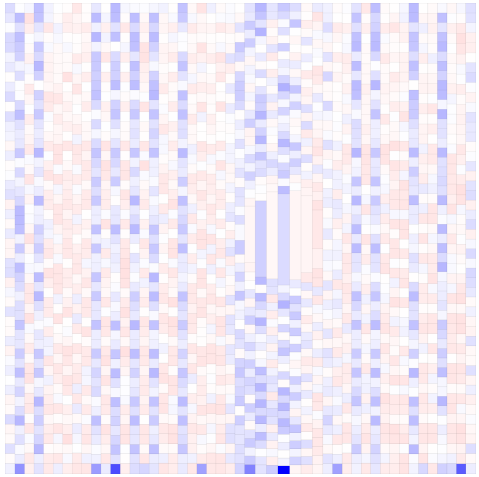
Uniform (17.5%)



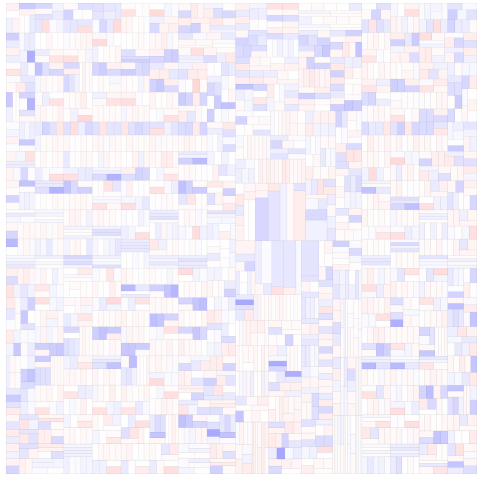
Rectilinear (15.1%)



$P \times Q$-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

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 $L(r) = \sum_{x \leq i \leq x', y \leq j \leq y'} A[i][j]$. 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 $Pr[i][j] = \sum_{i' \leq i, j' \leq j} A[i'][j']$. By convention $Pr[0][j] = Pr[i][0] = 0$.
We can now compute the load of rectangle $r = [x, y] \times [x', y']$ as $L(r) = Pr[x'][y'] - Pr[x-1][y'] - Pr[x'][y-1] + Pr[x-1][y-1]$.

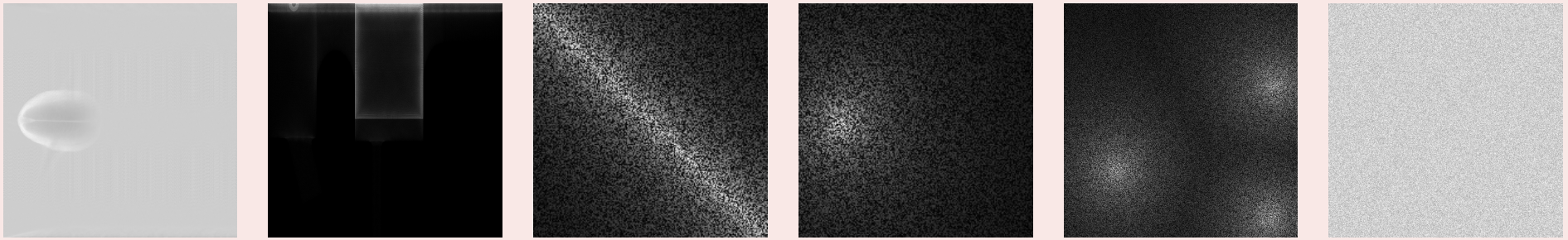
In One Dimension

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

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

Simulation Setting

Classes (Some inspired by [MS96])



Processors

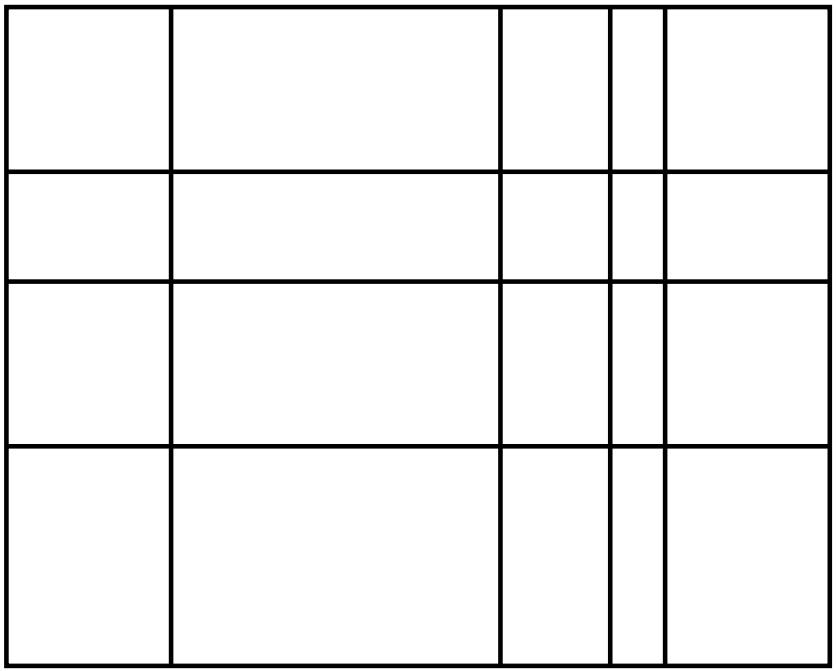
Simulation are perform with different number of processors: most squared numbers up to 10,000.

Metric

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

Outline of the Talk

Rectilinear Partitioning



Generalities

- The problem is NP-Hard.
- Approximation algorithms exist but are very slow.

RECT-NICOL [Nic94]

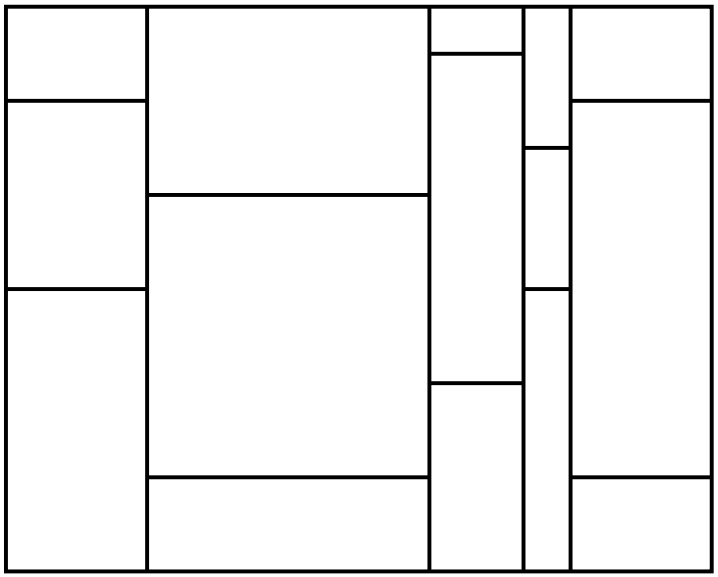
- An iterative heuristics.
- At each iteration the partition in one dimension is refined.

Complexity:

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

Outline of the Talk

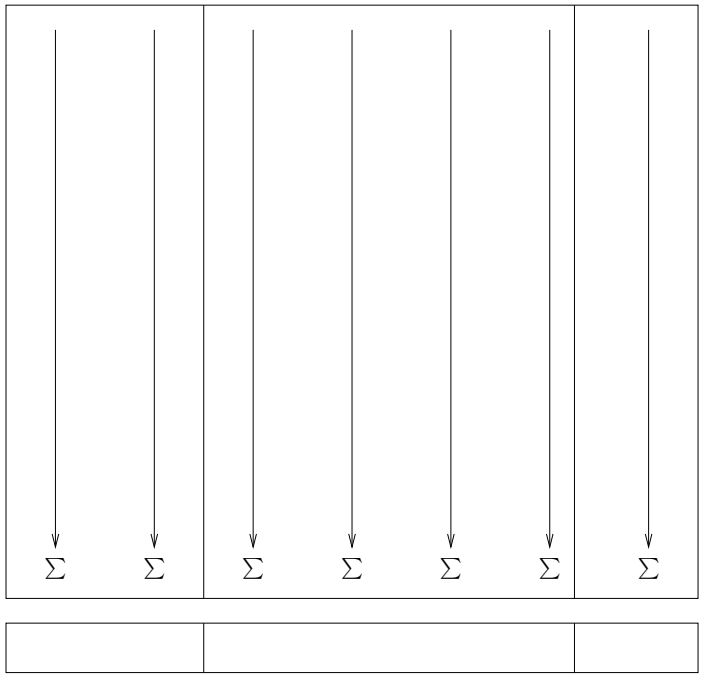
A $P \times Q$-way Jagged Heuristic



JAG-PQ-HEUR

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

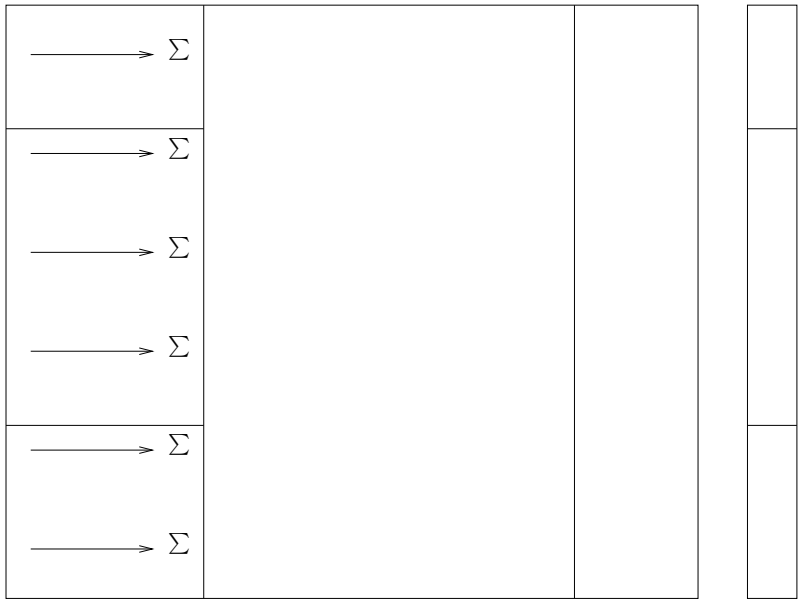
A $P \times Q$-way Jagged Heuristic



JAG-PQ-HEUR

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

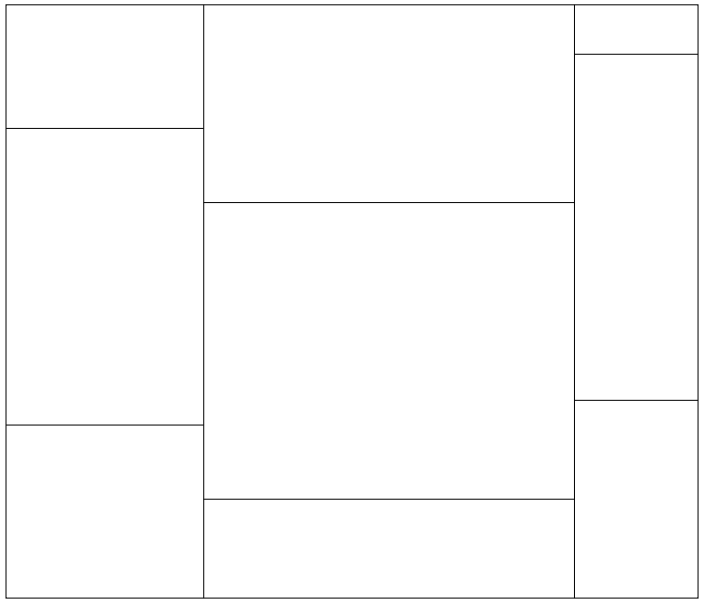
A $P \times Q$-way Jagged Heuristic



JAG-PQ-HEUR

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

A $P \times Q$-way Jagged Heuristic



JAG-PQ-HEUR

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

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

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

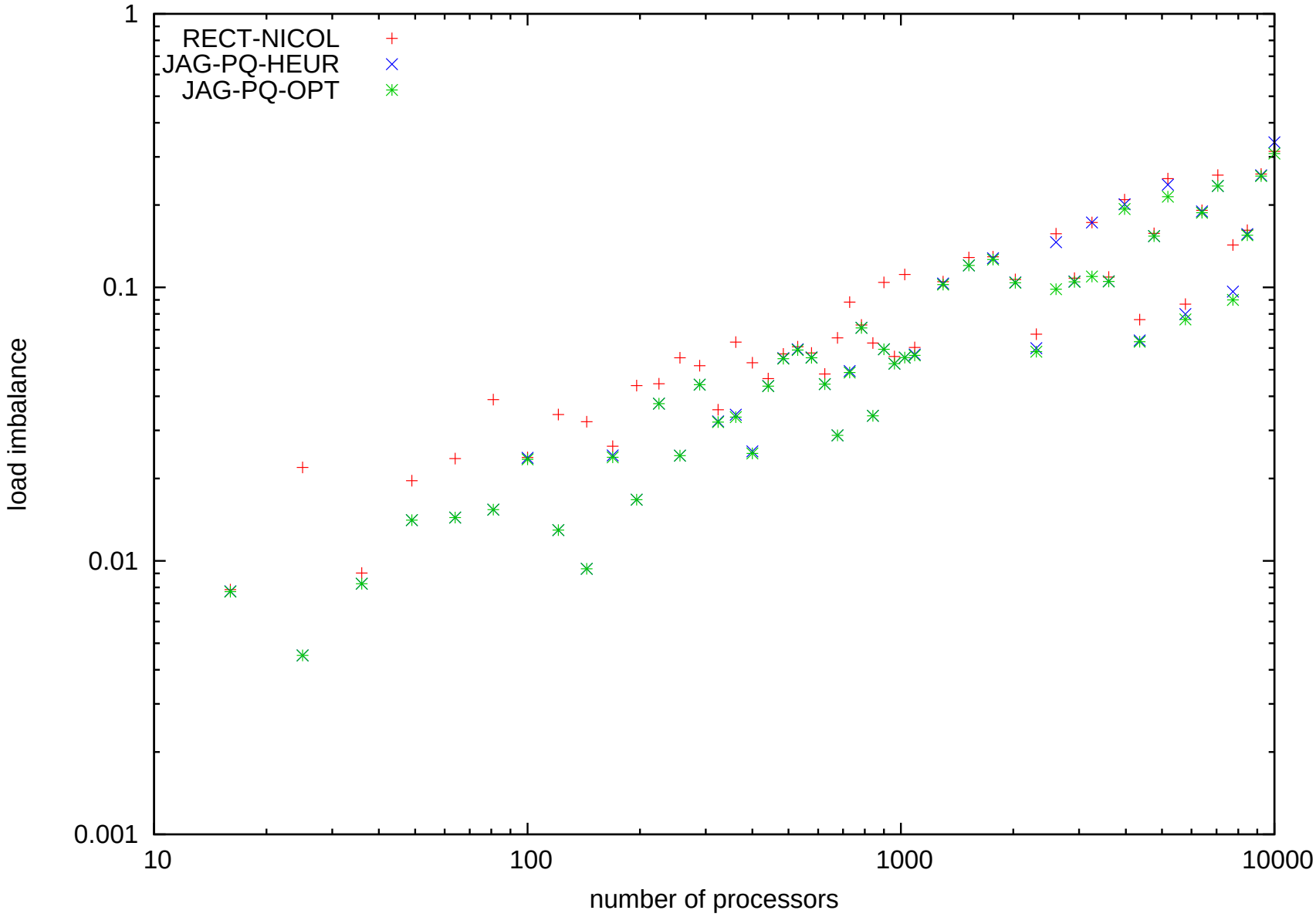
A Dynamic Programming Formulation

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

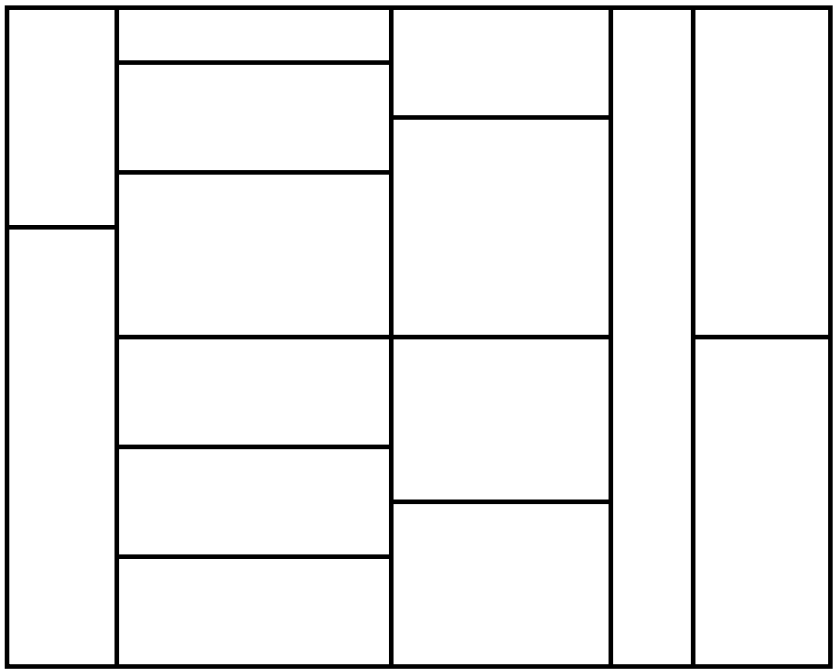
- $O(n_1 P)$ L_{max} functions to evaluate. (Each is $O(k)$.)
- $O(n_1^2)$ 1D functions to evaluate. (Each is $O((Q \log \frac{n_2}{Q})^2)$.)

(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)



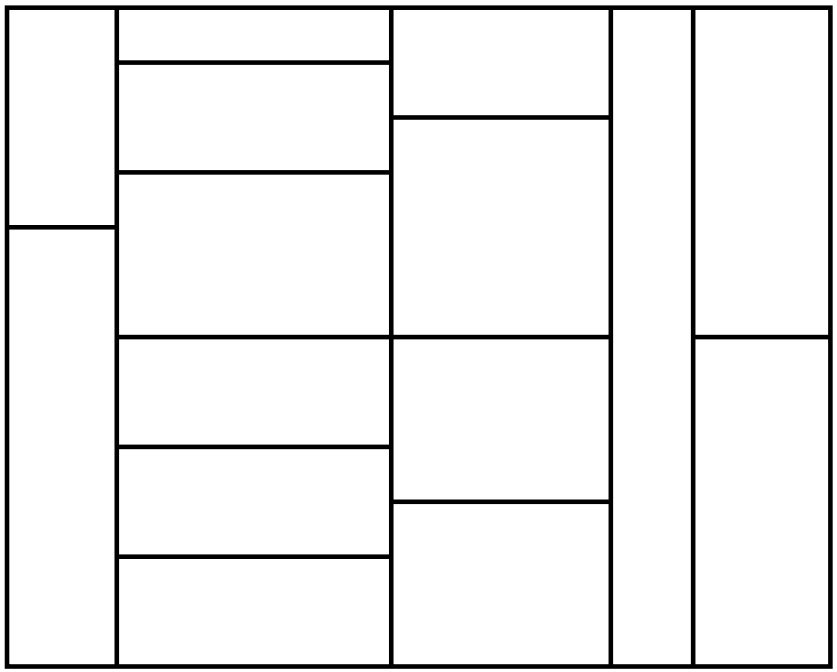
m-way jagged partitioning heuristics



JAG-M-HEUR

- Similar to JAG-PQ-HEUR.
- Cut in P stripes using an optimal 1D Algorithm.
- Distribute processors proportionally to the stripe's load.
- Compute a 1D partitioning of each stripe independently.

m-way jagged partitioning heuristics



JAG-M-HEUR

- Similar to JAG-PQ-HEUR.
- Cut in P stripes using an optimal 1D Algorithm.
- Distribute processors proportionally to the stripe's load.
- Compute a 1D partitioning of each stripe independently.

JAG-M-HEUR-PROBE

Partition all the stripes at once using a multiple 1D arrays partitioning algorithm [Fre92].

An optimal m-way partitioning JAG-M-OPT

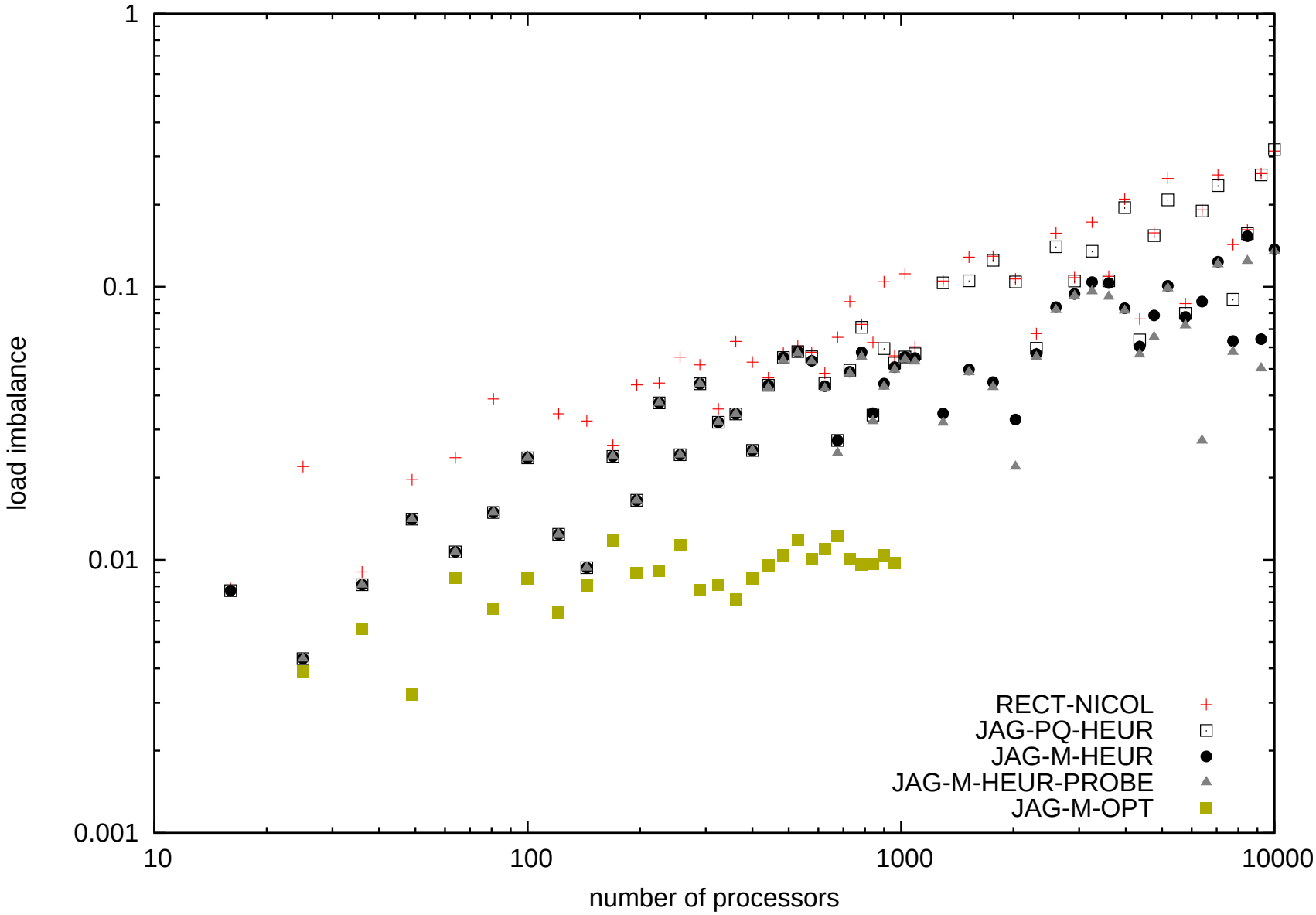
A Dynamic Programming Formulation

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

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

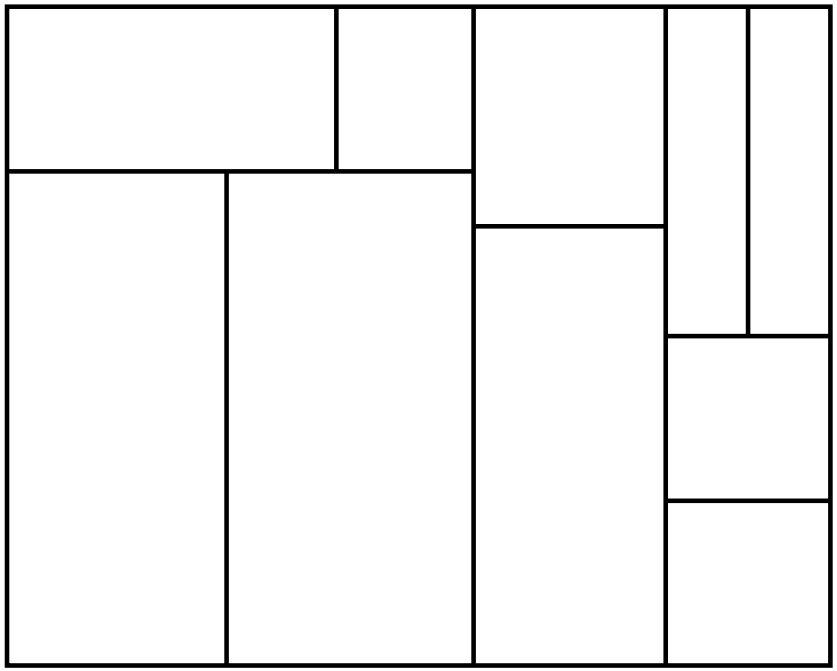
(The same kind of optimizations apply.)
For a 512x512 matrix on 1,000 processors. That's 512,000 + 262,144,000 values, if they are 64-bits, about 2GB (and takes 30 minutes).

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



Outline of the Talk

Heuristics for Hierarchical Bisection

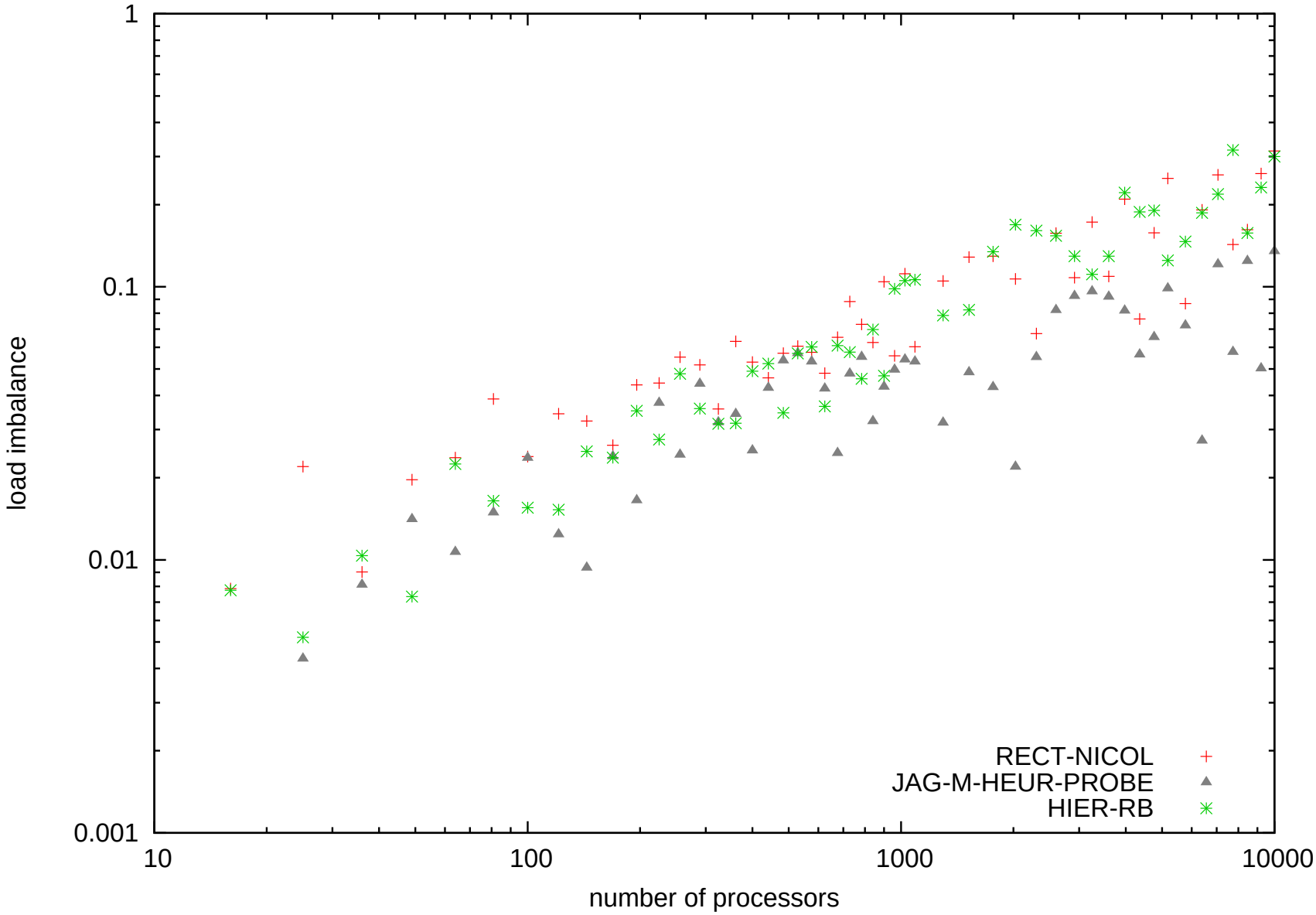


Recursive Bisection [BB87]: `HIER-RB`

- Cut to balance the load evenly.
- Allocate half the processors to each side.
- Cut the dimension 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

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

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

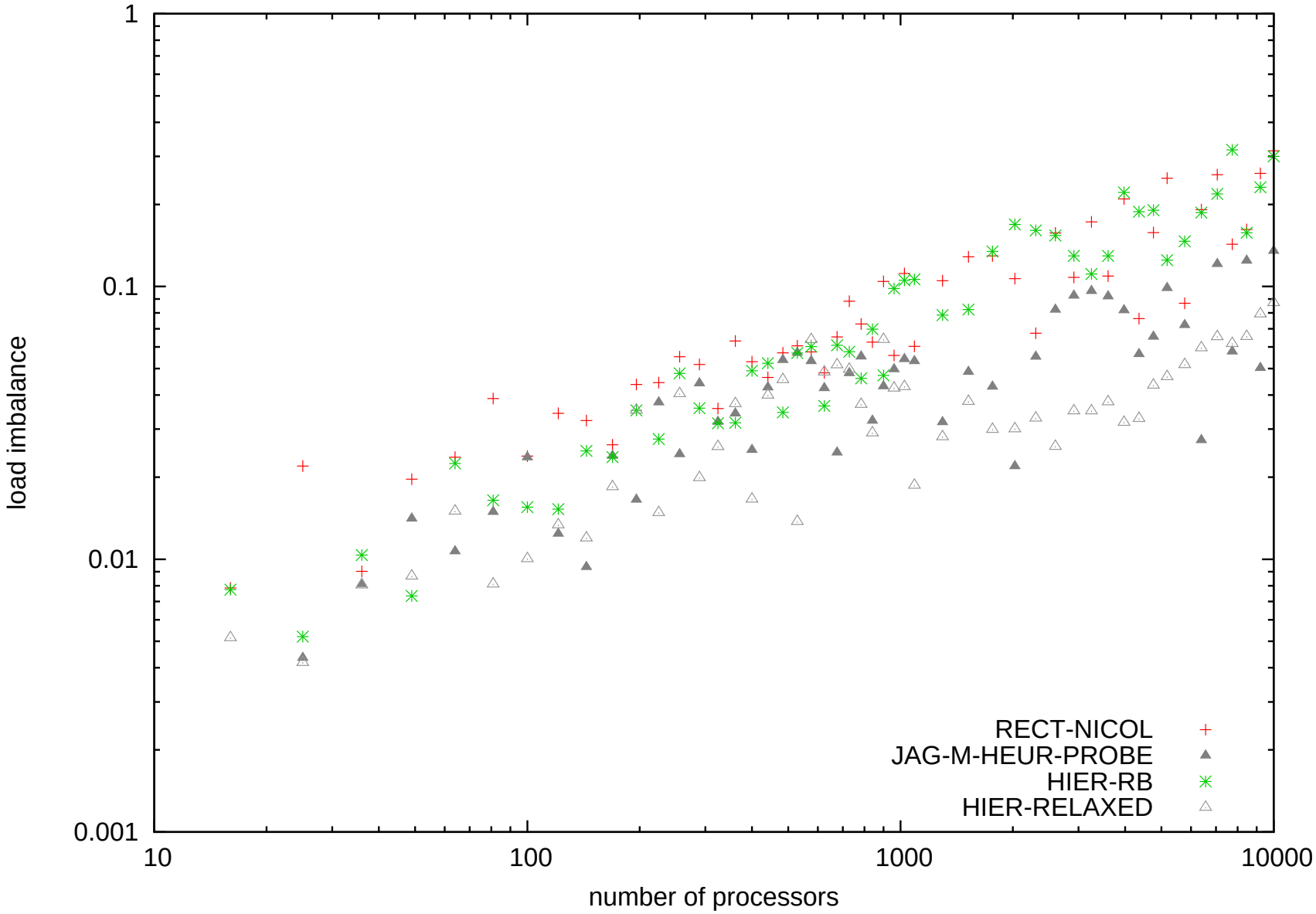
For a 512x512 matrix and 1000 processors, that's 68,719,476,736,000 values. On 64-bit values, that's 544TB.

The Relaxed Hierarchical Heuristic: `HIER-RELAXED`

Build the solution according to

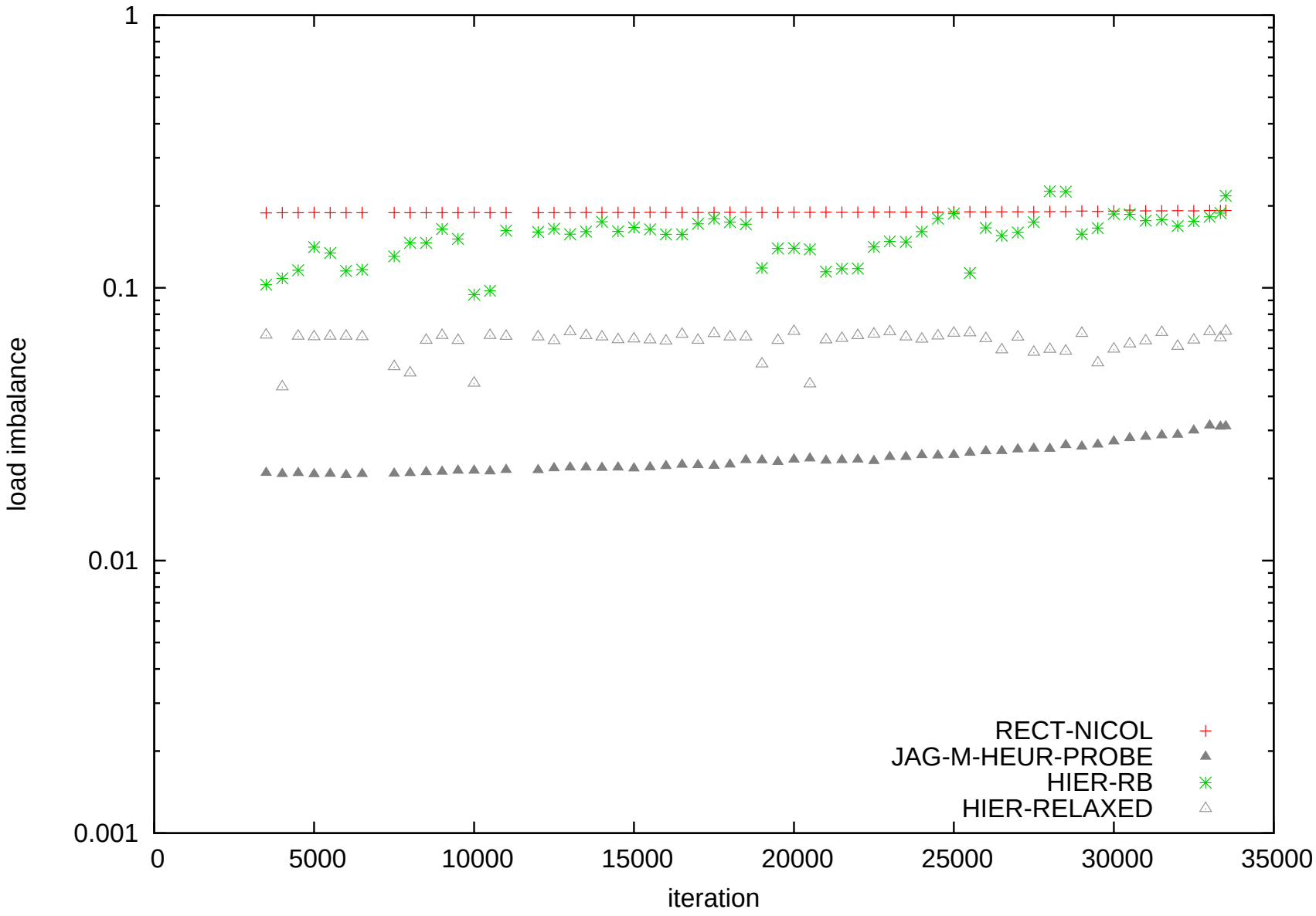
$$L_{max}(x_1, x_2, y_1, y_2, m) = \min_j \min(\min_x \max(\frac{L(x_1,x,y_1,y_2)}{j}, \frac{L(x+1,x_2,y_1,y_2)}{m-j}) , \min_y \max(\frac{L(x_1,x_2,y_1,y)}{j}, \frac{L(x_1,x_2,y+1,y_2)}{m-j}))$$

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

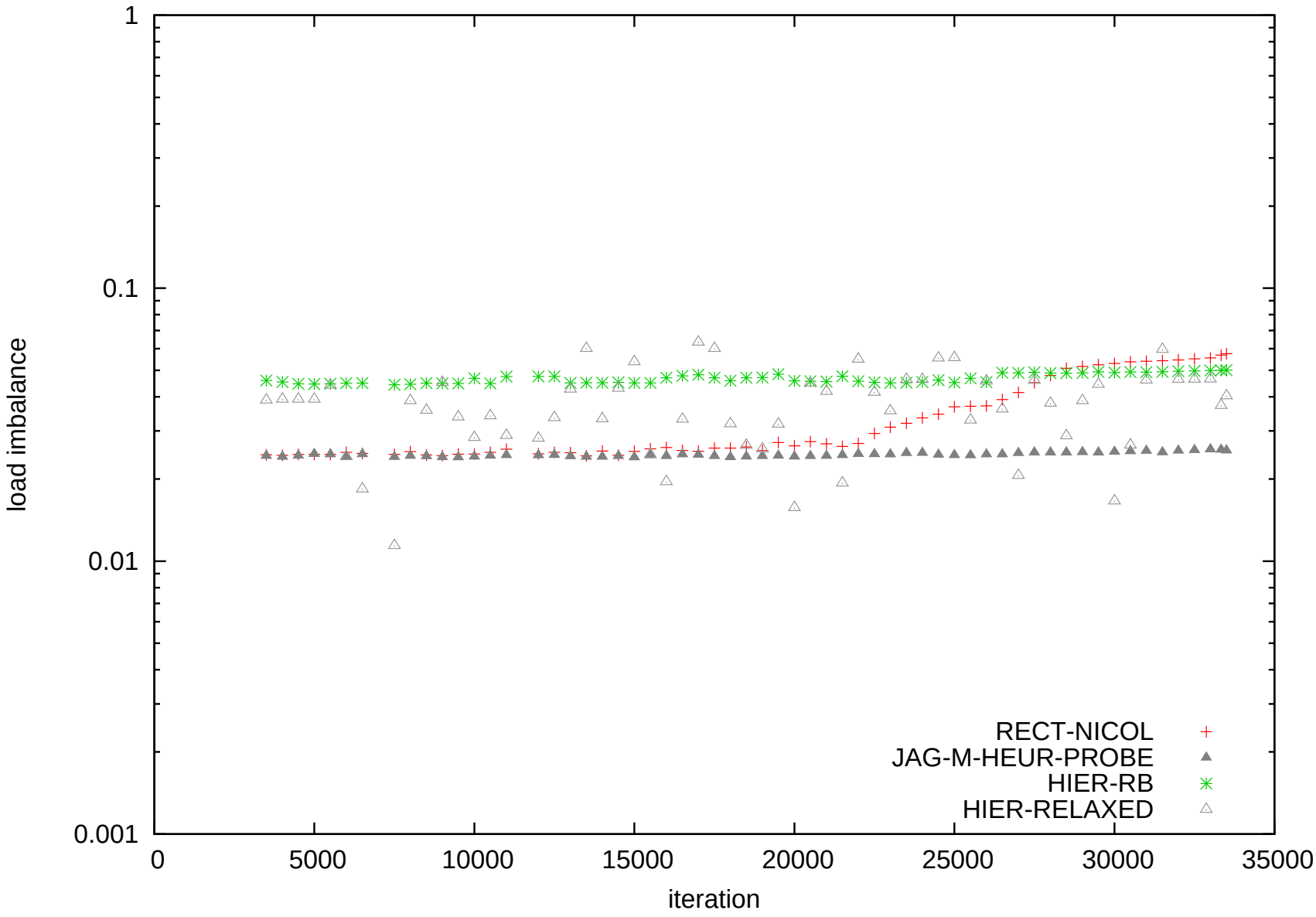


Outline of the Talk

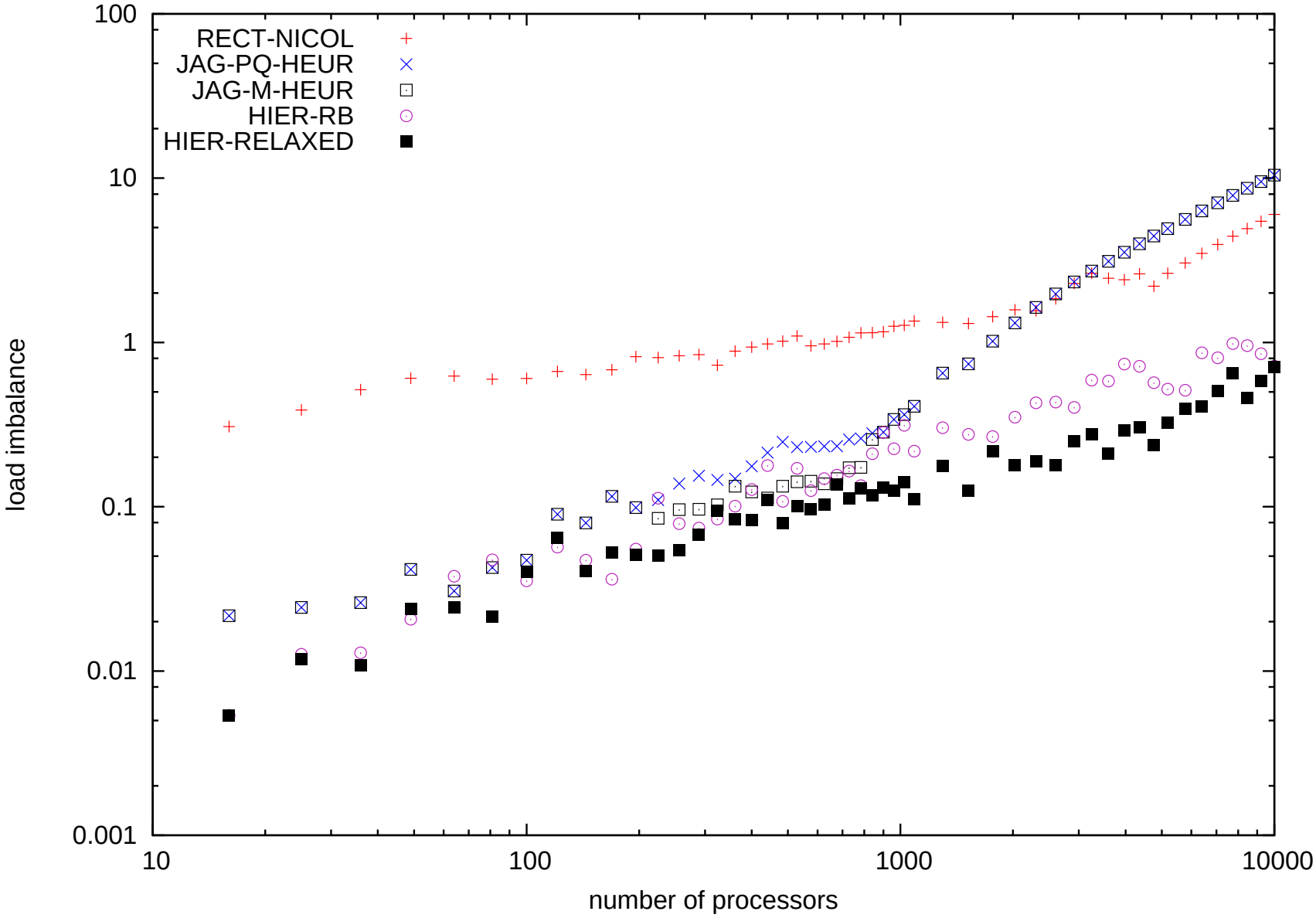
Performance Over the Execution of PIC-MAG (m =6400)



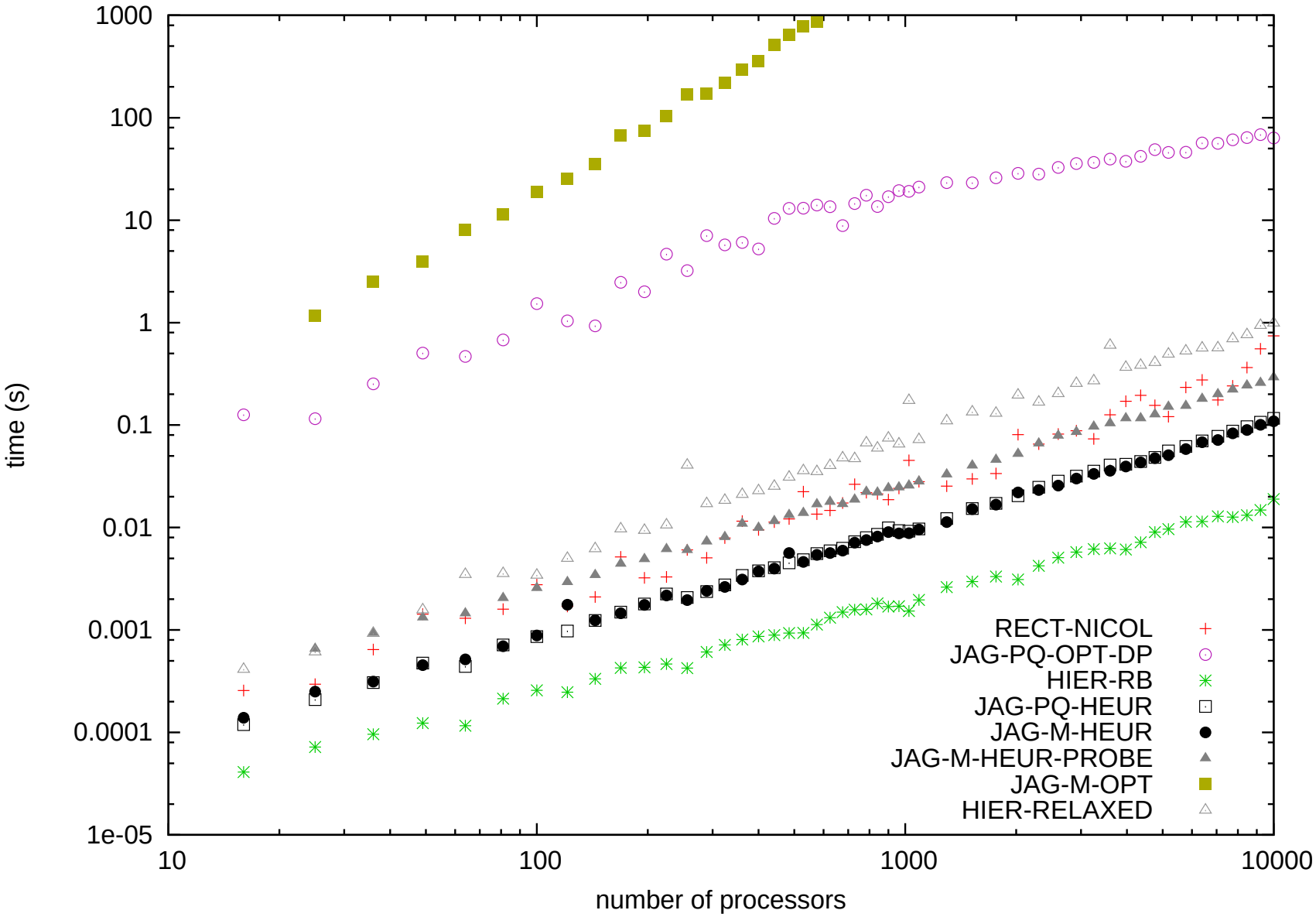
Relaxed Hierarchical Might Be Unstable (m =400)



Sparsity (SLAC)



Runtime on PIC-MAG (it=30000)



What should I use?

Dense instances

- JAG-M-HEUR-PROBE and HIER-RELAXED dominates. (Best of two?)
- But HIER-RELAXED is unstable: it gives very different solutions when run on similar instances.

Sparse instances

- Jagged partitions can reach a worse case scenario.
- Hierarchical partitions get better results: HIER-RELAXED is the best.

Runtime (on a 514x514 matrix with 1024 processors)

- HIER-RB one milliseconds
- JAG-PQ-HEUR, JAG-M-HEUR: 10 milliseconds.
- HIER-RELAXED, RECT-NICOL, JAG-M-HEUR-PROBE: 50 milliseconds.
- JAG-M-OPT: hours.

What did I left out?

More details in our Technical Report (arXiv 1104.2566)

- Guarantees for most heuristics (approximation ratio).
- *m*-way jagged admits optimal algorithms for fixed column cut and for fixed processor distribution.
- Multi-level partitioning can be used to achieve better solutions.

What did I left out?

More details in our Technical Report (arXiv 1104.2566)

- Guarantees for most heuristics (approximation ratio).
- *m*-way jagged admits optimal algorithms for fixed column cut and for fixed processor distribution.
- Multi-level partitioning can be used to achieve better solutions.

Will these algorithms help your application?

A sequential tool is available! Check it out at
`http://bmi.osu.edu/hpc/software/spart/`

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/`, `http://bmi.osu.edu/~umit` or `http://bmi.osu.edu/hpc/software/spart/`

Research at HPC lab is funded by

DAGSI The Dayton Area Graduate Studies Institute
DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA
INL Idaho National Laboratory
NATIONAL CANCER INSTITUTE
NATIONAL INSTITUTES OF HEALTH
NSF
Sandia National Laboratories


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


Greg N. Frederickson.
Optimal algorithms for partitioning trees and locating p-centers in trees.
Technical Report CSD-TR-1029, Purdue University, 1990, revised 1992.


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.