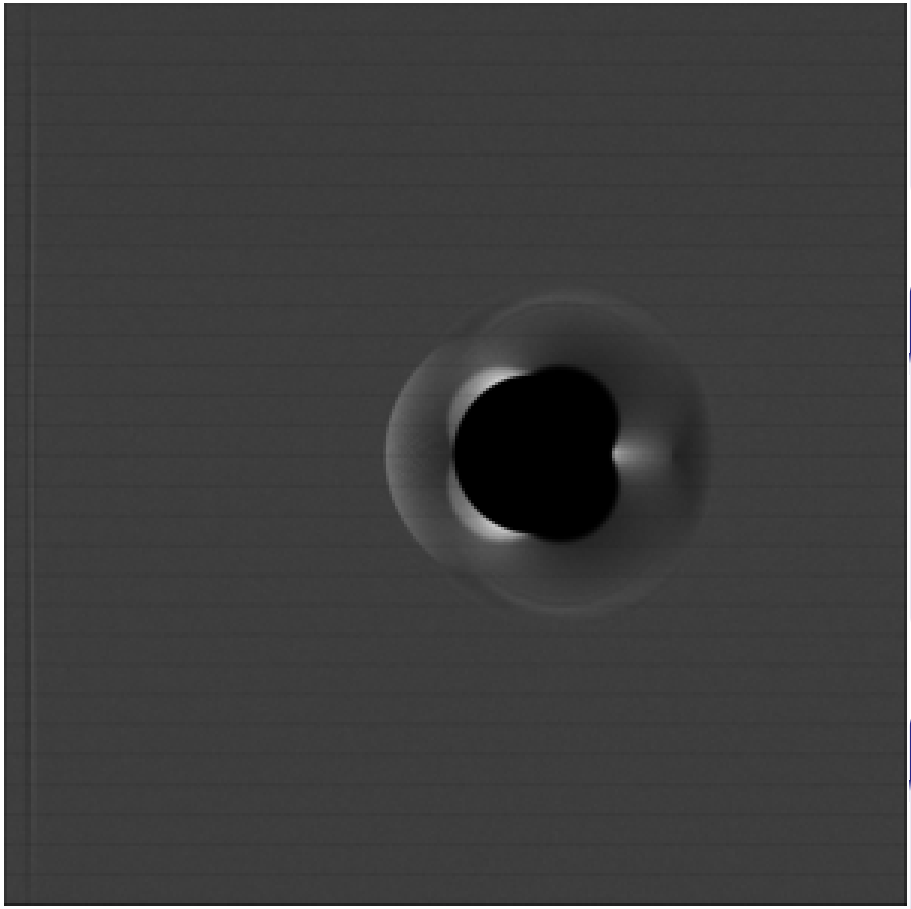
Spatial partitioning scheme - the one dimension case

Erdeniz Ozgun Bas, **Erik Saule**, Umit V. Catalyurek

Department of Biomedical Informatics, The Ohio State University
{erdeniz,esaule,umit}@bmi.osu.edu

HPC lab weekly meeting - March 16, 2010

A load distribution problem



Load matrix

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

Applications

- particle in cell
- sparse matrices
- direct volume rendering

Metrics

- Load balance
- Communication
- Stability

How to solve the 2d problem ?

Calling on 1d partitioning

PxQ way jagged partitioning algorithm partitions the array in P vertical stripes. Each one is partitioned in Q parts.
A heuristic way of doing it cuts the in vertical stripes by aggregating the rows into a 1d problem. And each stripes is partitioned using a 1D algorithm. ($P + 1$ calls to 1D)
A more clever algorithm uses binary searches to find more interesting vertical cutting points. (and does $P \log n$ calls to 1D)

Let's take some numbers

For a bluegene machine that's $65K = 2^8 \times 2^8$ processors.
For a internet.mtx (from UFMC) that's $120K \times 120K = 2^{17} \times 2^{17}$
heuristic is 257 1d calls and the more clever is $17 \times 2^8 = 4352$ 1d calls.

1D algorithms must be good!

Outline of the Talk

Notation

Task

In all the rest of the presentation we will consider an array A of size n : $A[1], \ldots, A[n]$.
A is given to the algorithms through a prefix sum array Pr where $Pr[0] = 0$ so that $\sum_{i=begin}^{end} A[i] = Pr[end] - Pr[begin - 1]$.
Computing the prefix sum array is never taken into account in complexity and timings.

Processors

The array will be partitioned in m intervals.
We assume that $m \leq n$

Outline of the Talk

Parametric Search

Principle

- Try to build a solution of bottleneck value B.
- Greedily load the processors up to B.
- If all the array is allocated, B is feasible.
- Otherwise, it is not.

Probe

procedure `probe`(B,m, Pr, n)
    $s[0] = 0$
    for $j = 1$ to m **do**
        $B_{pre} \leftarrow Pr[s[j-1]] + B$
        $s[j] \leftarrow BSearch(Pr, s[j-1], n, B_{pre})$;
    return $B_{pre} \geq W_{tot}$

Complexity: $O(m \log n)$

Probe by [Han, IPL 92]

Improved version in $O(m \log \frac{n}{m})$

procedure probe(B,m, Pr, n)
  Let $inc = \frac{n}{m}$
  $step \leftarrow inc$;
  $s[0] \leftarrow 0$
  for $j = 1$ to m **do**
    $B_{pre} \leftarrow Pr[s[j-1]] + B$
    while $step \leq n$ AND $Pr[step] < B_{pre}$ **do**
      $step \leftarrow \min(step + inc, n)$;
    $s[j] \leftarrow BSearch(Pr, step - inc, step, B_{pre})$;
  return $B_{pre} \geq W_{tot}$

Nicol Algorithm [Nicol, JPDC 1994]

Principle

For processor j only two intervals are worthwhile starting at $i[j-1]$ up to

- minimum $i[j]$ where Probe is true, if j is the bottleneck
- maximum $i[j]$ where Probe is false, if j is not the bottleneck

Nicol Minus

procedure `Nicol`(m, Pr, n)
  $i[0] \leftarrow 1$
  for $j = 1$ to $m-1$ **do**
    $i[j] \leftarrow \arg\min_{i[j-1]<i\leq n} Probe(Pr[i] - Pr[i[j-1]-1])$ is true
    $B[j] \leftarrow Pr[i] - Pr[i[j-1]-1]$
  $B[m] \leftarrow Pr[n] - Pr[i[m-1]-1]$
  return $\min_j B[j]$

Complexity : $O(m^2 \log n \log \frac{n}{m})$ but can be improved to $O((m \log \frac{n}{m})^2)$

Nicol with Dynamic Bound Checking [Pinar, JPDC 2004]

Monotonicity of Probe

If $Probe(B_0)$ is true then $\forall B \geq B_0, Probe(B)$ is true.
If $Probe(B_0)$ is false then $\forall B \leq B_0, Probe(B)$ is false.

Nicol

An adaptation of Nicol Minus which recalls the value of previous call to probe.

Complexity : $O(m^2 \log n \log \frac{n}{m})$ but can be improved to $O((m \log \frac{n}{m})^2)$

Nicol with Separator Index Bounding [Pinar, JPDC 2004]

Idea

Reuse the cuts of previous calls to probe.
Let $s_0[j]$ be the cuts computed by $Probe(B_0)$ and $s_1[j]$ be the cuts computed by $Probe(B_1)$. If $B_0 \leq B_1$ then $\forall j, s_0[j] \leq s_1[j]$.

Nicol Plus

Inside *Probe*, restrict the binary search to $[SL[b] : SH[b]]$ where SL (resp. SH) are the cuts of a previous unsuccessful (resp. successful) call to probe.

Complexity : $O((m \log \frac{n}{m})^2)$ and
$O(m \log n + A_{max}(m \log m + m \log(\frac{A_{max}}{A_{avg}})))$

Benchmark

Random Arrays

Generated uniformly with number of tasks from 10^5 to 10^8. Each size is repeted 10 times.

Sparse Matrices

Downloaded from UFL sparse matrix collection.
Each matrix is transformed into two 1d instances by counting the number of element per row and column

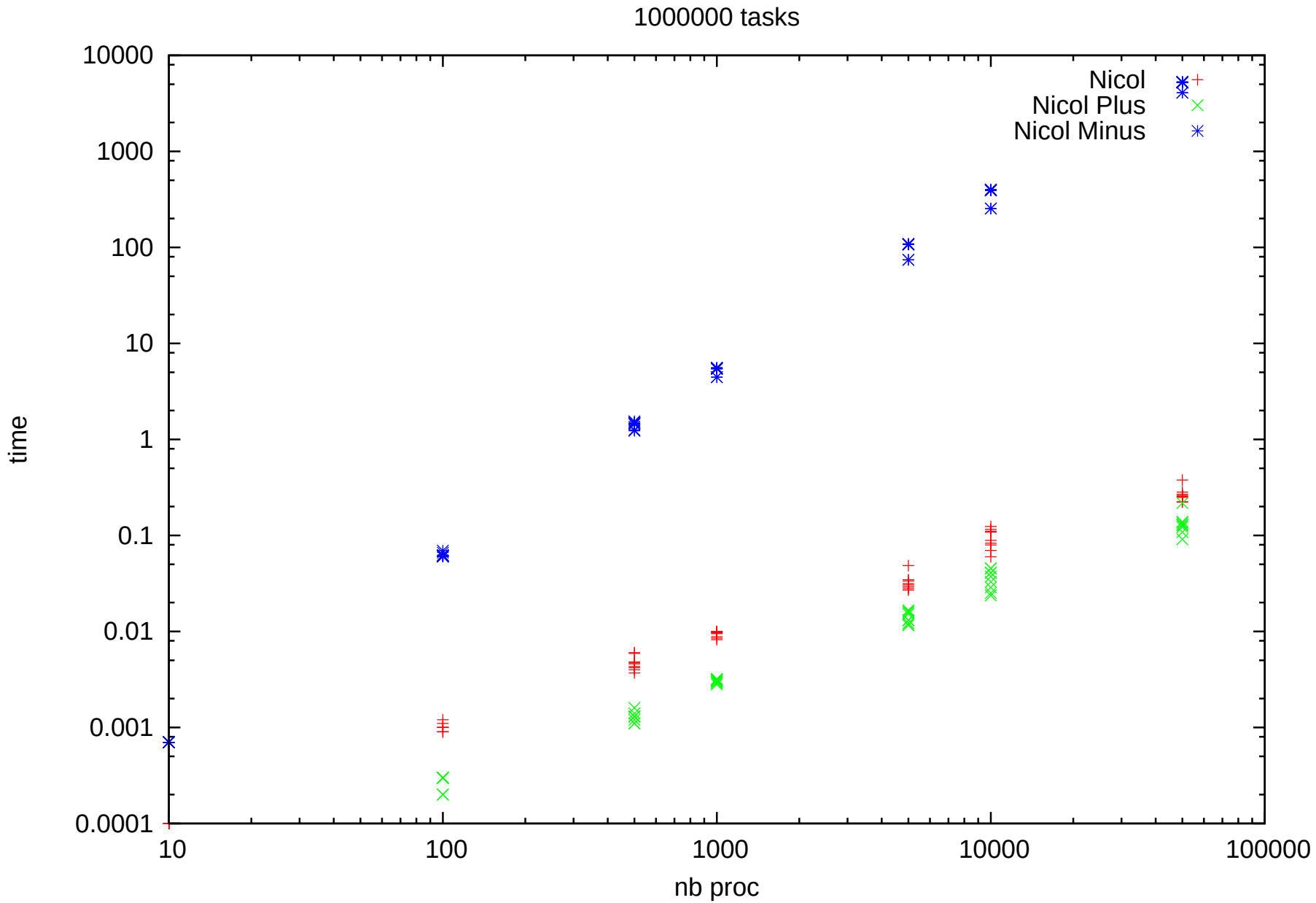
Processors

m is taken between 10 and 5.10^4

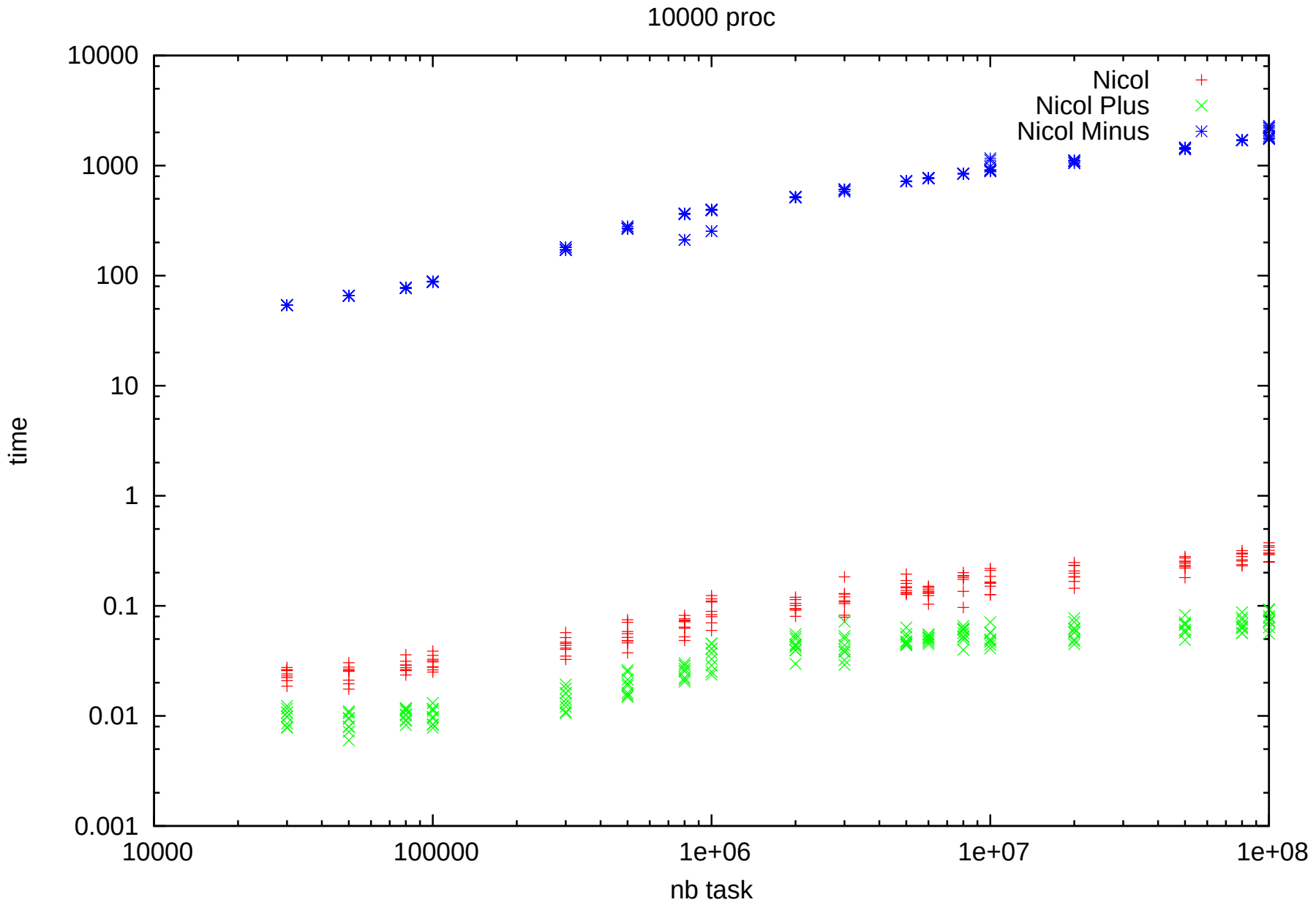
Variations

Each measure is repeted 5 times. std dev and variance are not reported but very small.

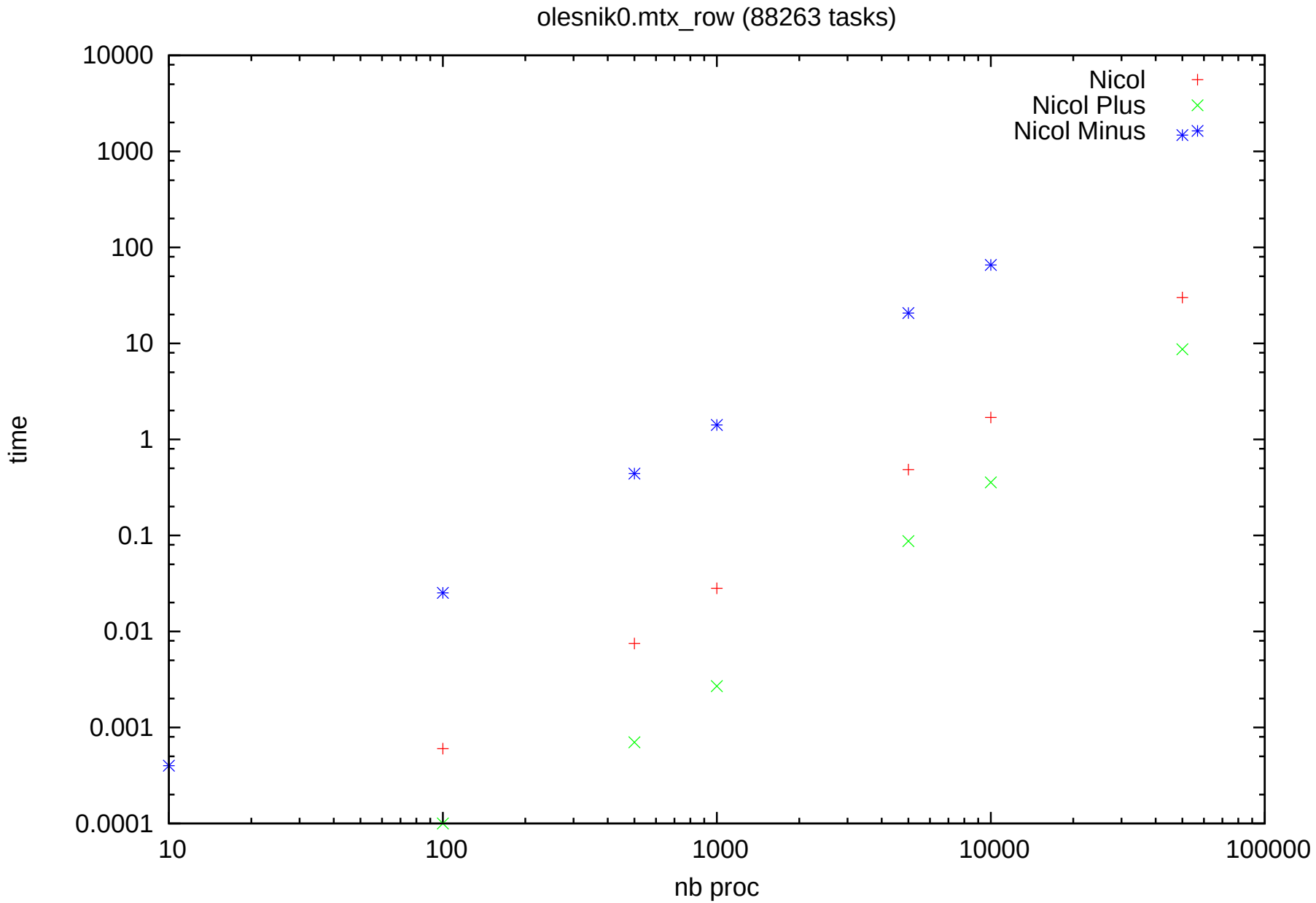
Random arrays



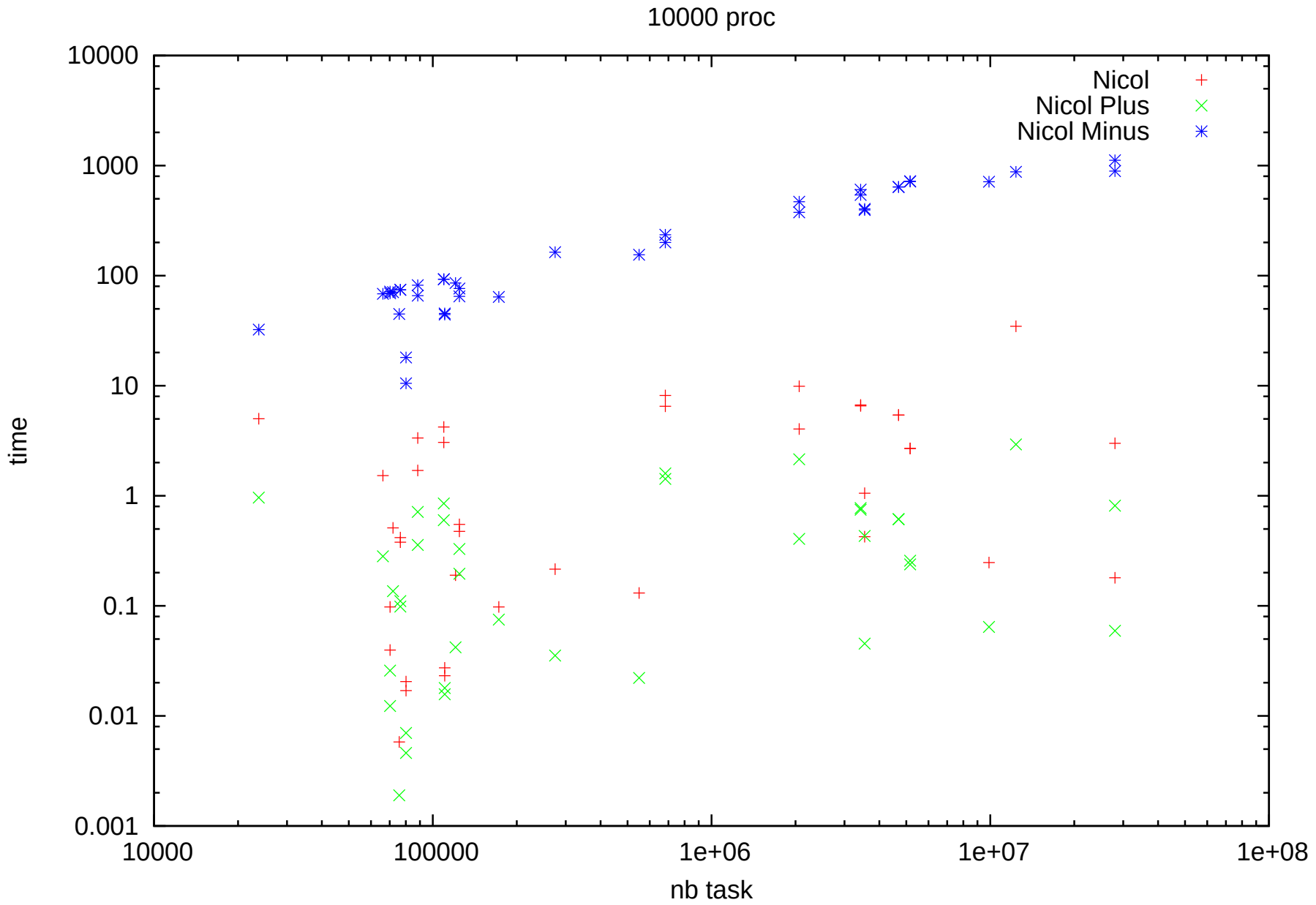
Random arrays



UFL matrices



UFL matrices



Outline of the Talk

Recursive Bisection [Bokhari, IEEE TC 1987]

Algorithm

Idea: recursively cut the array in two

procedure `RecursiveBisection`$(Pr, low, high, m)$
  if $m = 1$ **then**
    return $Pr[high] - Pr[low - 1]$
  Let $(c1, v1) = cutEvenly(Pr, low, high, \lfloor m/2 \rfloor, \lceil m/2 \rceil)$
  Let $(c2, v2) = cutEvenly(Pr, low, high, \lceil m/2 \rceil, \lfloor m/2 \rfloor)$
  if $v1 < v2$ **then**
    return $RB(Pr, low, c1, \lfloor m/2 \rfloor) + RB(Pr, c1 + 1, high, \lceil m/2 \rceil)$
  else
    return $RB(Pr, low, c2, \lceil m/2 \rceil) + RB(Pr, c2 + 1, high, \lfloor m/2 \rfloor)$

Analysis

- Performance : $B_{RB} \leq \frac{\sum_i A[i]}{m} + \frac{m-1}{m} \max_i A[i] \leq 2B_{opt}$
- Complexity: $O(m \log n)$

Greedy Bisection [???]

Algorithm

Idea: Greedily cut the largest array in two

procedure `GreedyBisection`$(Pr, low, high, m)$
  Let H be an empty heap.
  $H.push([low; high], Pr[high] - Pr[low - 1])$
  while $H.size() \neq m$ **do**
    Let $[a; b] = h.popMax()$
    Let $(c, v) = cutEvenly(Pr, a, b, 1, 1)$
    $H.push([a; c], Pr[c] - Pr[a - 1])$
    $H.push([c + 1; b], Pr[b] - Pr[c])$

Analysis

- Performance : $B_{GB} \leq 2\frac{\sum_i A[i]}{m+1} + \frac{(m-1)}{m+1} \max_i A[i] \leq 3B_{opt}$.
- Complexity: $O(m \log n)$.

Direct Cut [Miguet, HPCN 1997]

Algorithm

Idea: cut every $\frac{\sum_i A[i]}{m}$.

procedure `Direct Cut`$(Pr, low, high, m)$

Let $avg = \frac{Pr[high]-Pr[low-1]}{m}$ and $inc = \frac{high-low}{m}$

$cut_0 \leftarrow low$; $step \leftarrow inc$; $cost \leftarrow 0$

for $j = 1$ **to** $m-1$ **do**

while $Pr[step] < j * avg$ **do**

$step \leftarrow step + inc$

$cut_j \leftarrow BinarySearch^{\geq}(Pr, step - inc, step, j * avg)$

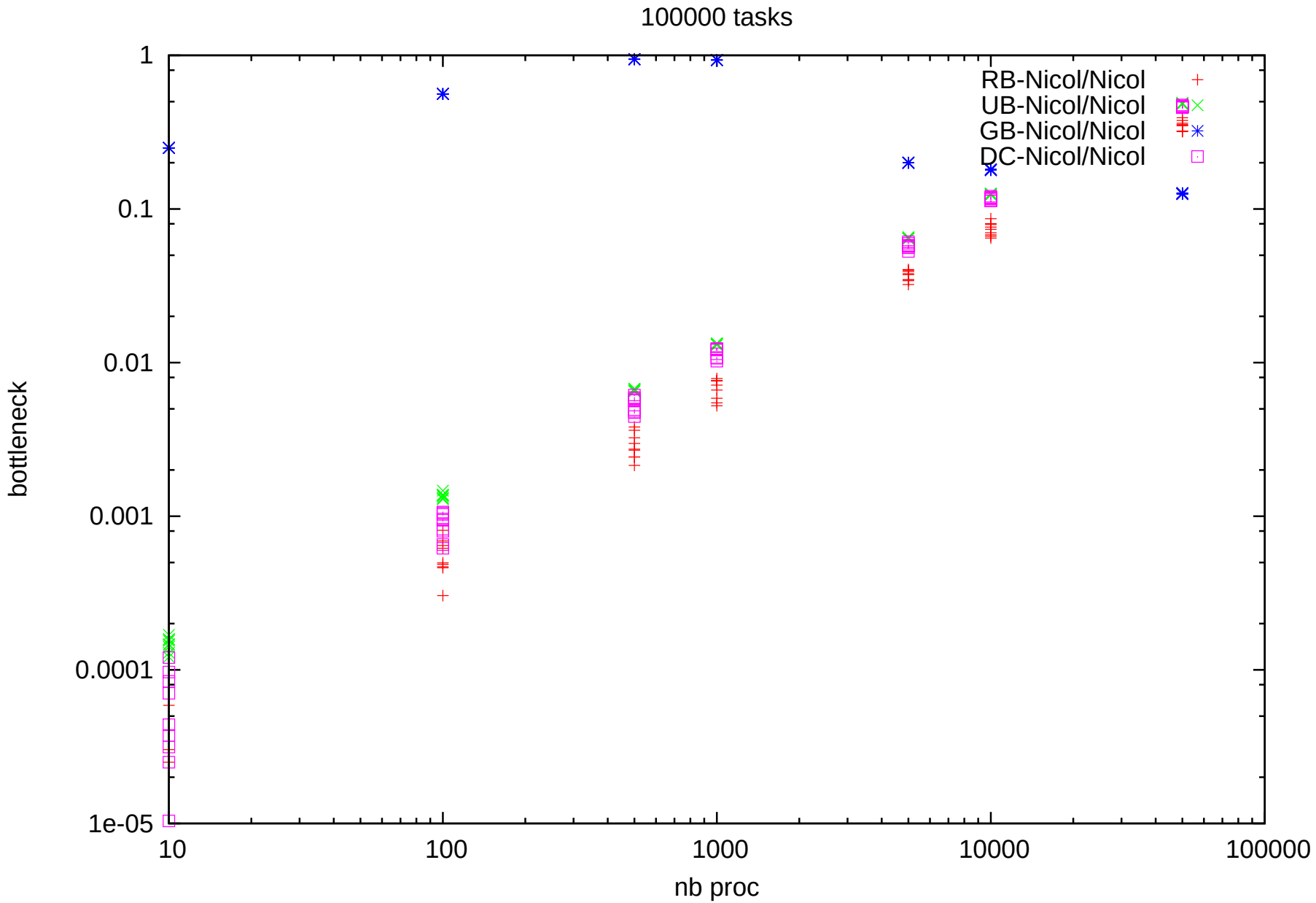
$cost \leftarrow max(cost, Pr[cut_j] - Pr[cut_{j-1}])$

return *cost*

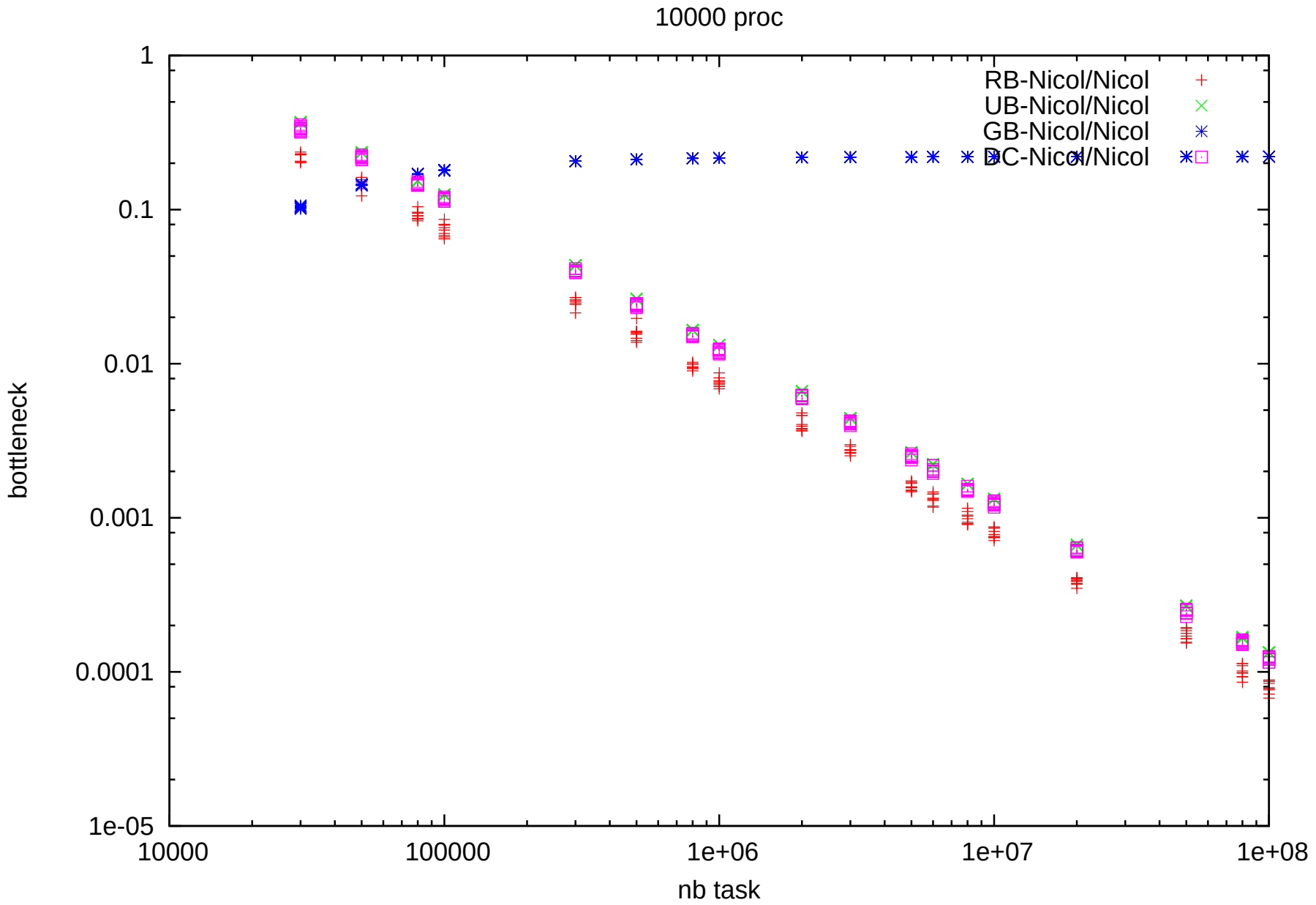
Analysis

- Performance : $B_{DC} \leq \frac{\sum_i A[i]}{m} + \max_i A[i]$
- Complexity: $O(m \log \frac{n}{m})$

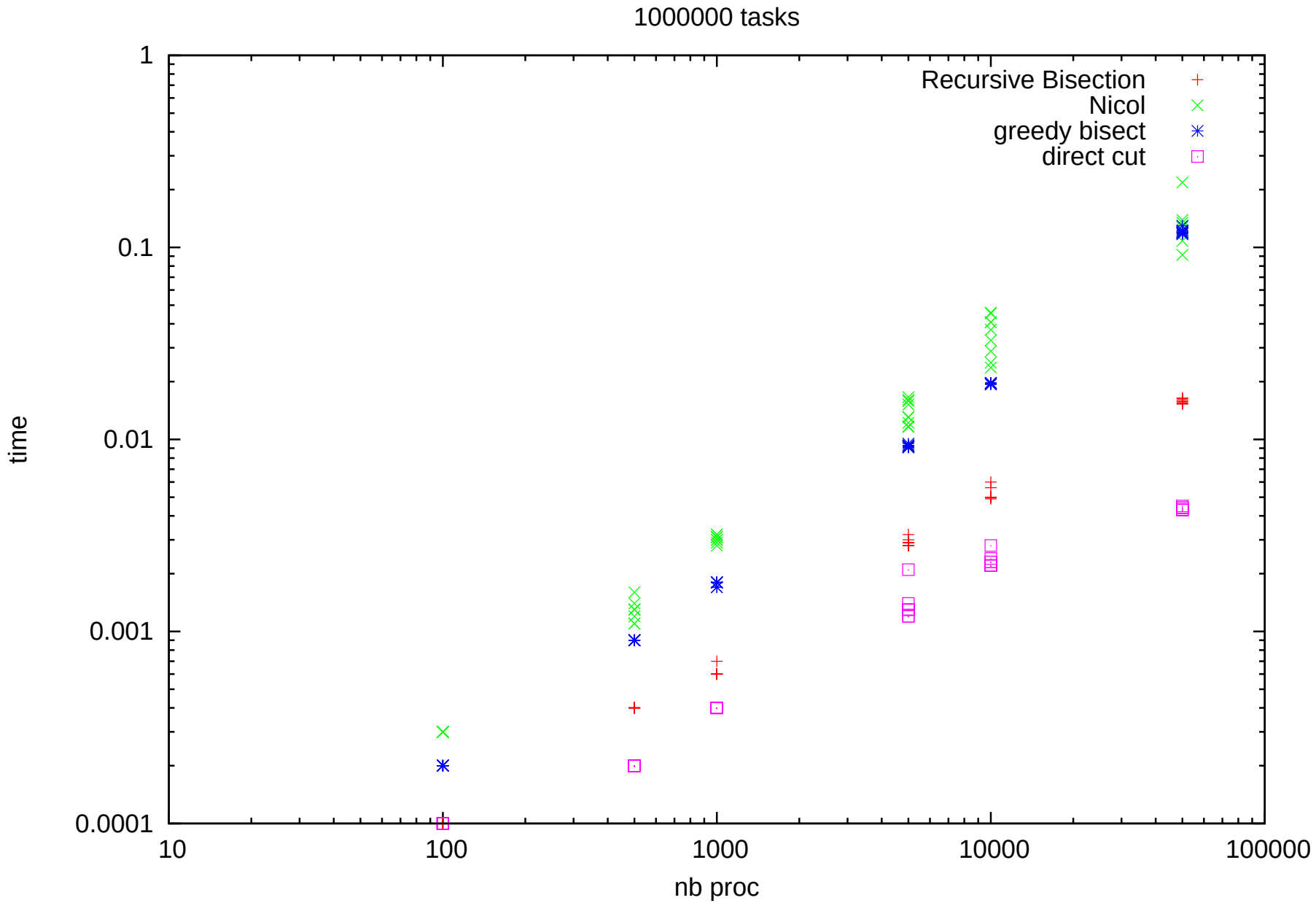
Random arrays - Error



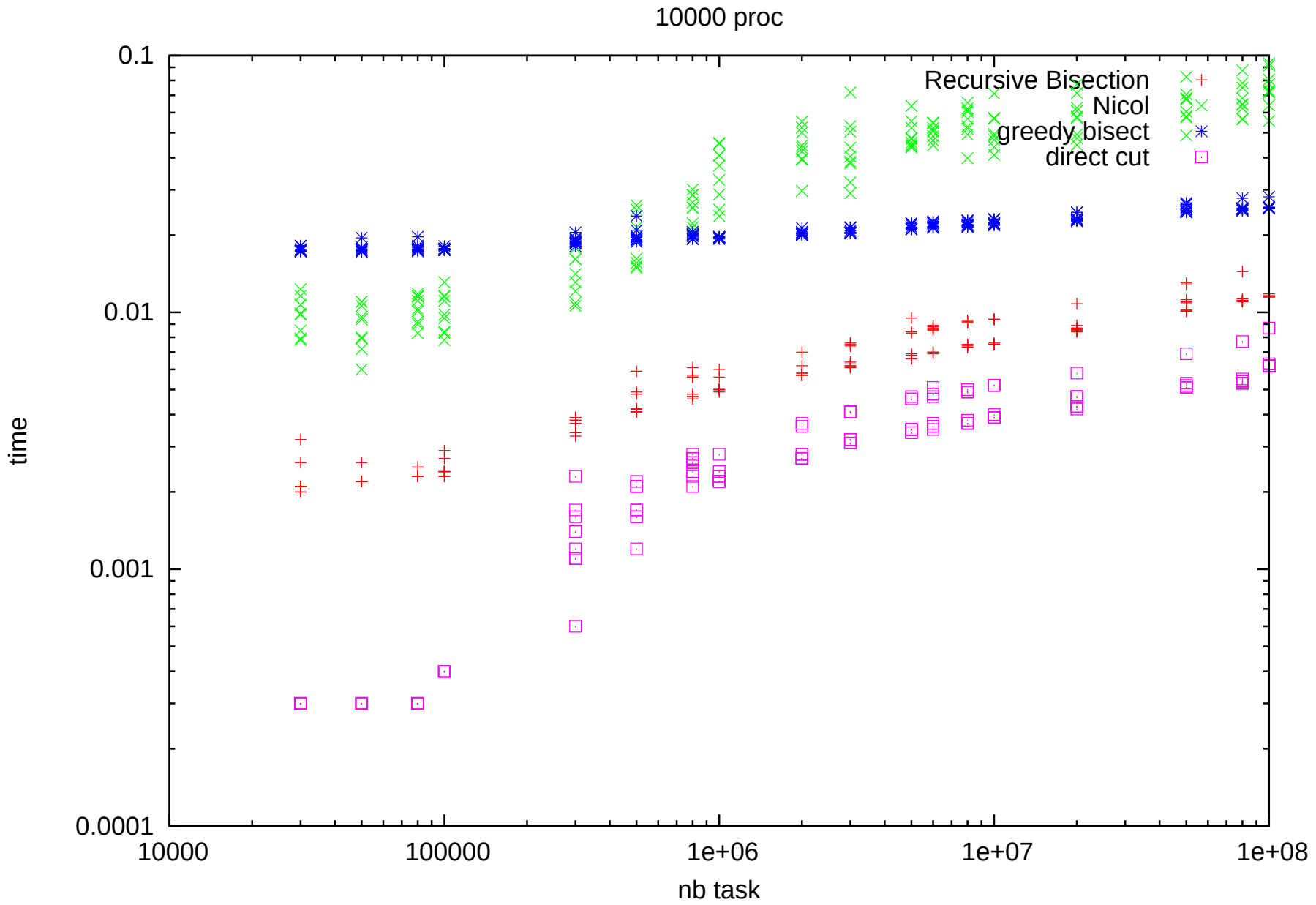
Random arrays - Error



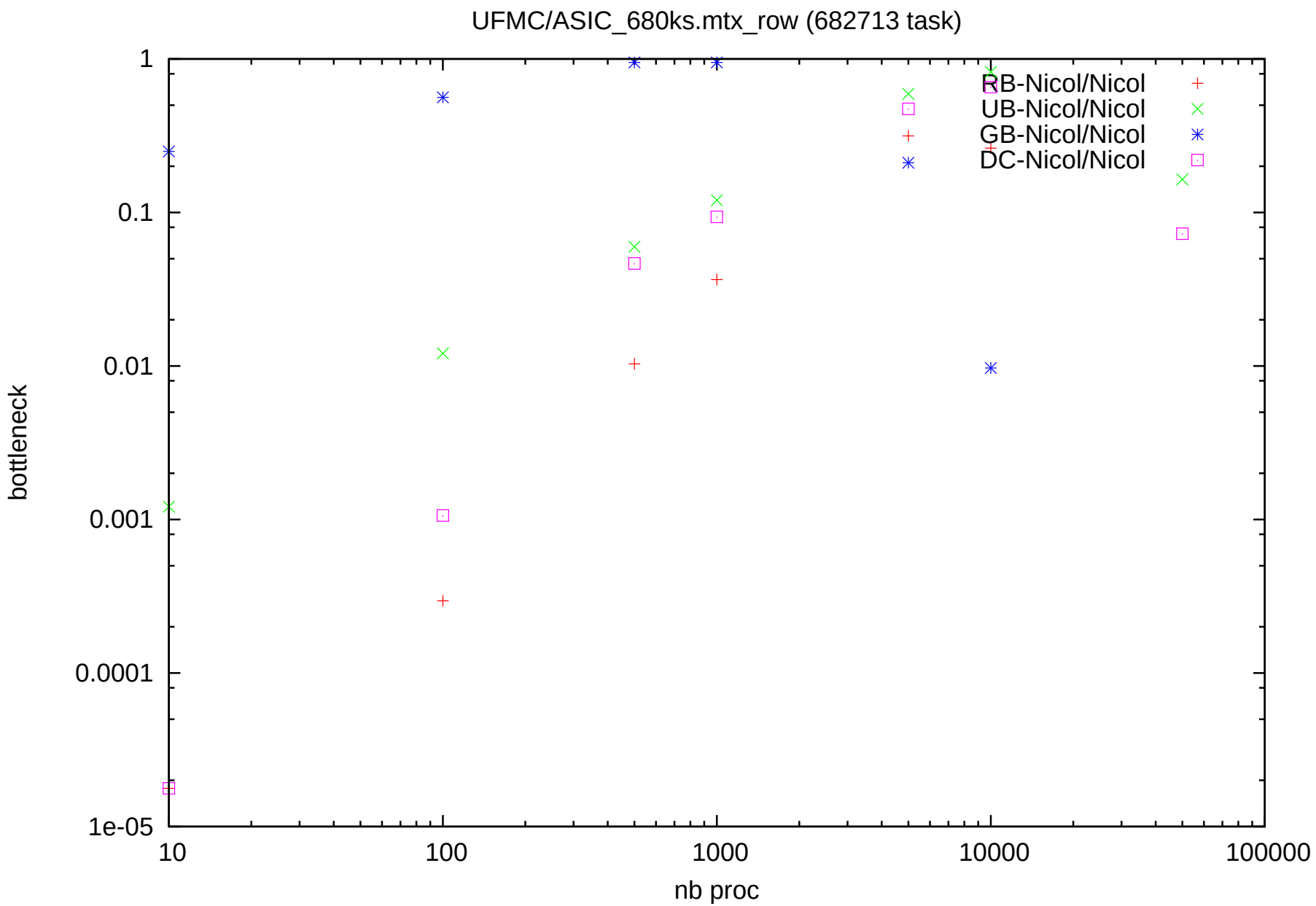
Random arrays - Time



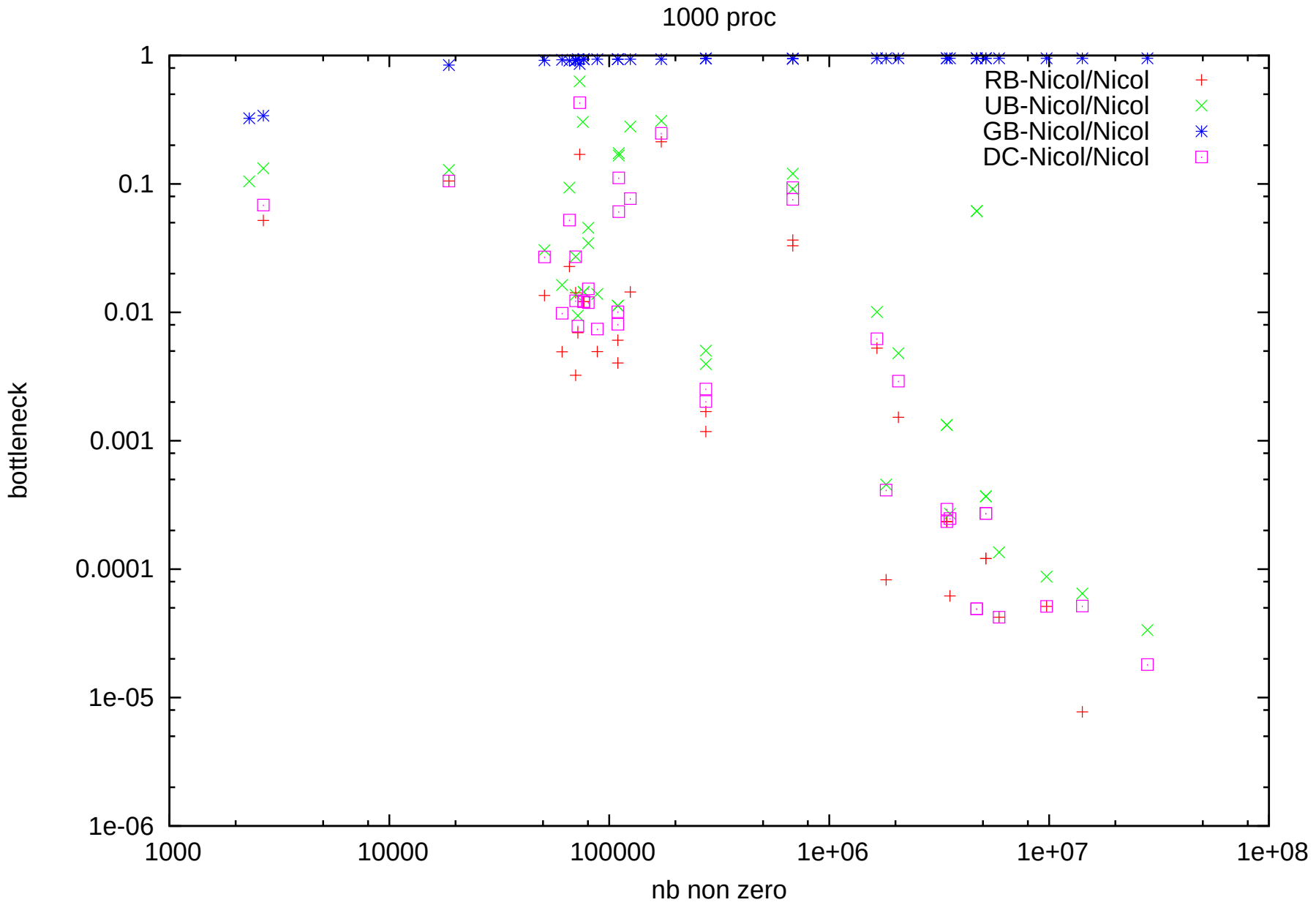
Random arrays - Time



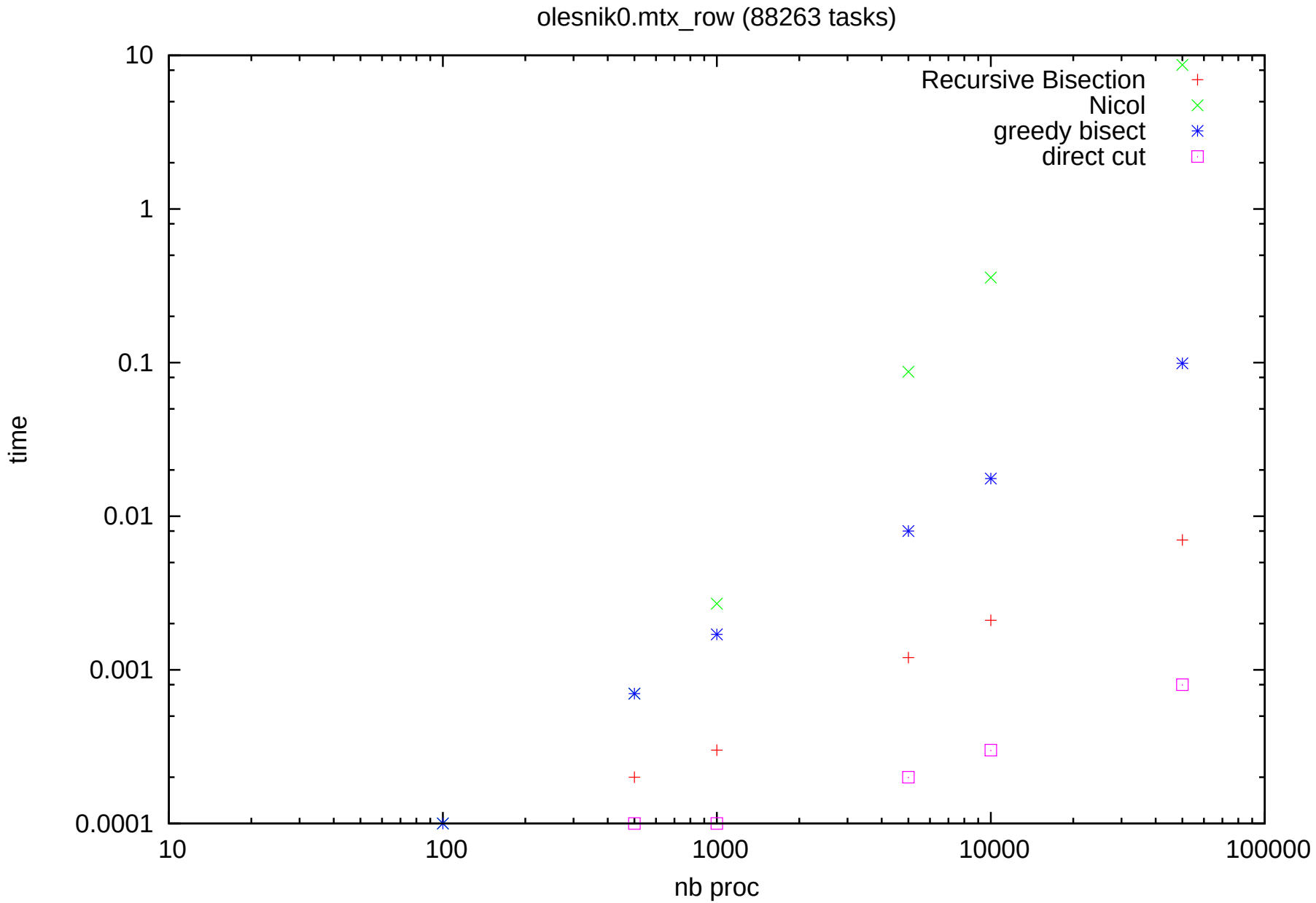
UFL matrices - Error



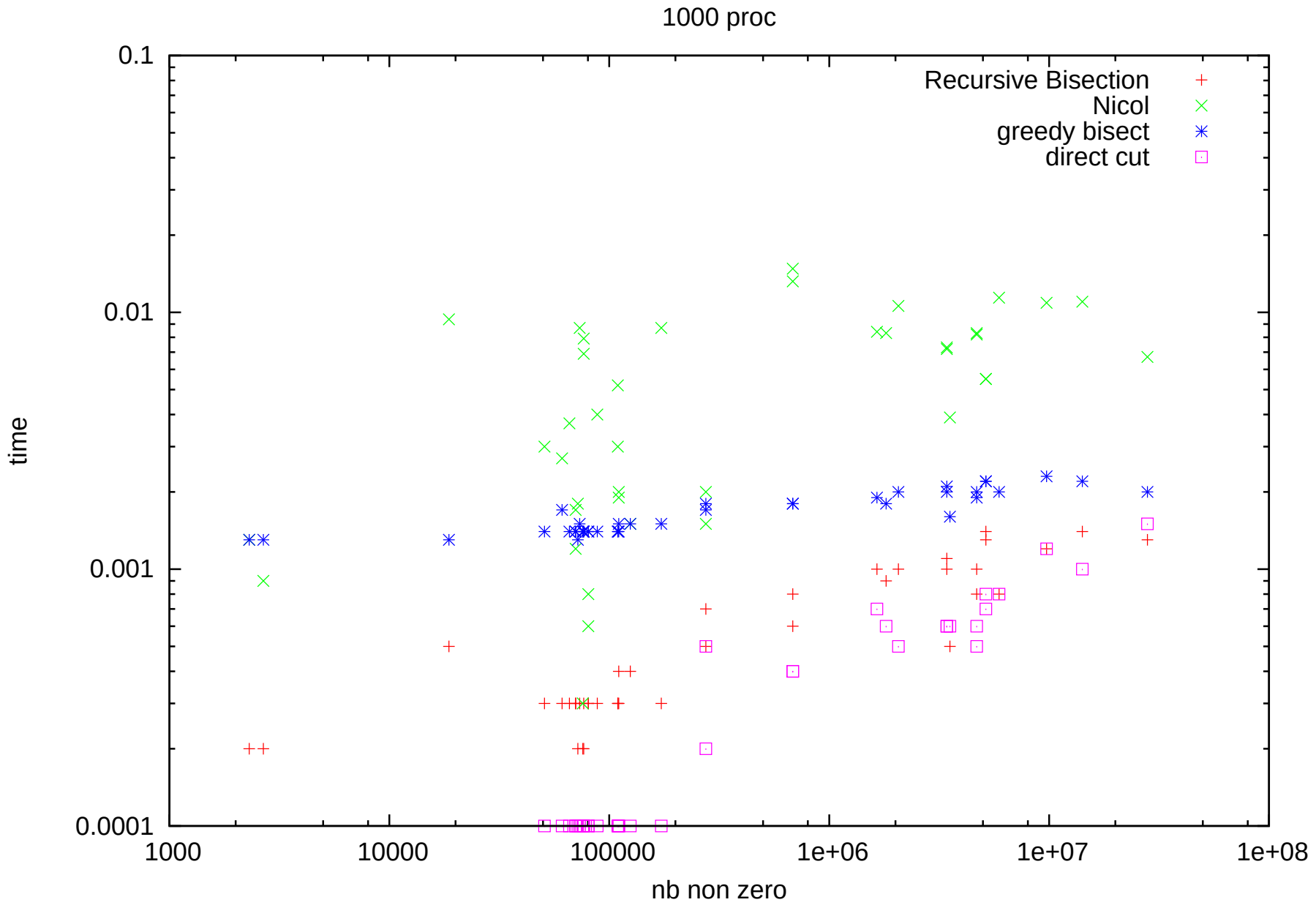
UFL matrices - Error



UFL matrices - Time



UFL matrices - Time



Outline of the Talk

Conclusion

On optimality

Nicol's algorithm can be largely improved by removing useless computation. Even if complexity (big O notation) didnot change, the speedup is significant (2 orders of magnitude)

On heuristic

Heuristic can be even faster (between 1 and 2 orders of magnitude) by losing little on the load balance. RB gets better load balance than DC but is also slower.

Non reported data

- Similar results on homa instances
- An improvement on Direct Cut has been made with little changes
- Considering only non zero as number of task does not change anything

Future Works : Going 2/3D

NB: similar results on homa's data set

Rectilinear Partitioning

- Is NP-Complete in 2D and 3D
- Nicol JPDC 94: describes a way to generate then
- Several approximation algorithms exist

Jagged Partitioning

- Easy heuristics
- Two optimal P-way x Q-way algorithms are known
- a optimal P processor algorithm can be designed

Recursive Bisection approaches

- Bokhari 88 : describes how to do recursive bisection
- The optimal recursive bisection can be computed by DP