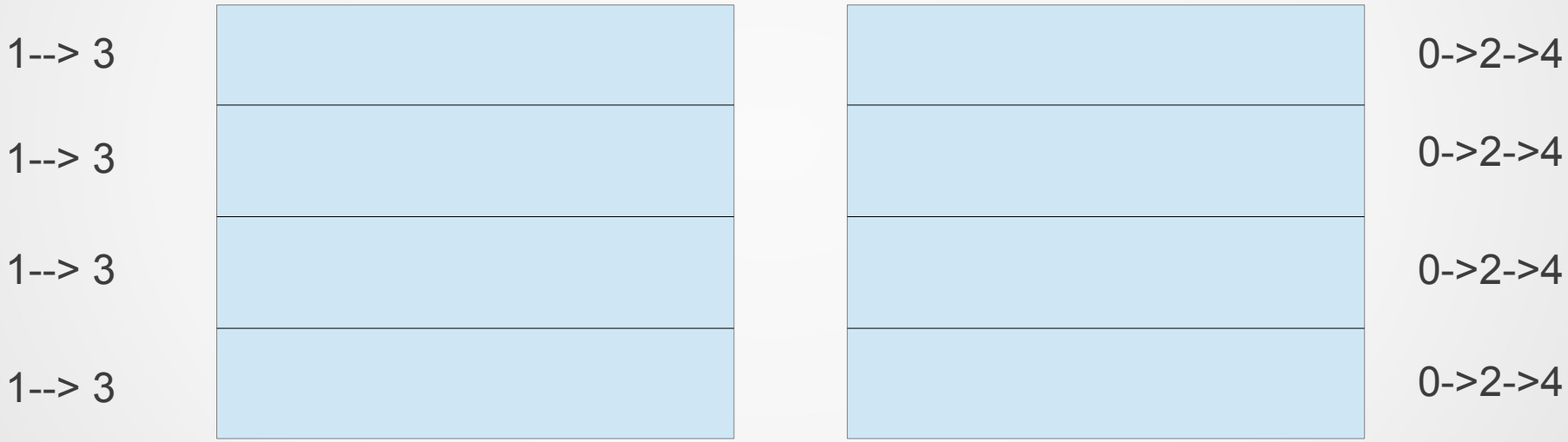
Automatic Parallelization & Manual Parallelization with Open MP

Shrikant Vinchurkar
Mayank Chaudhary

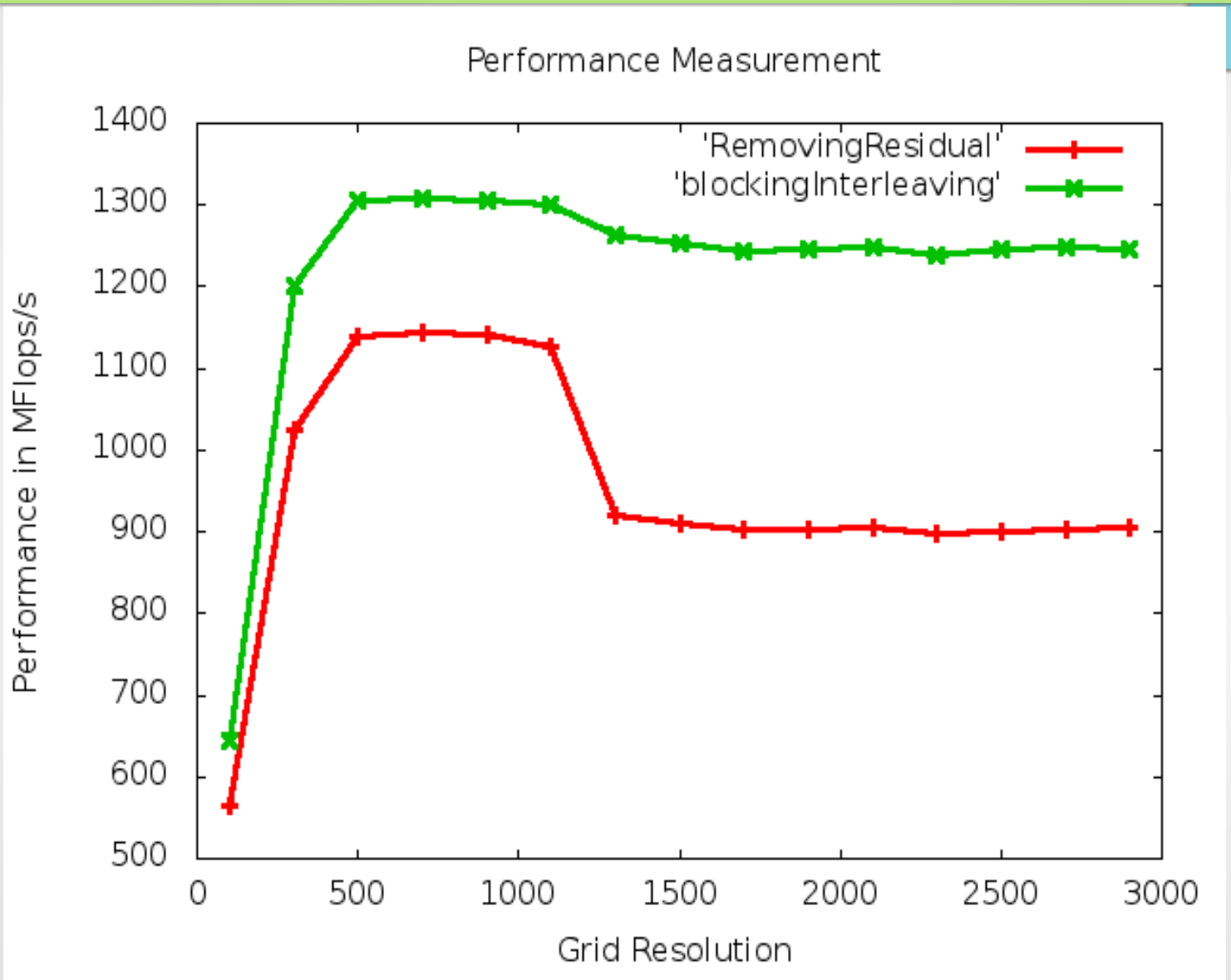
Results from blocking

- Horizontal blocking into 4 blocks
- Interleaving of iterations
- Removed copy operation in relax_jacobi
- Time reduced from 11 sec to 8.76
- We are continuing OpenMP parallelization with this code

Interleaving approach



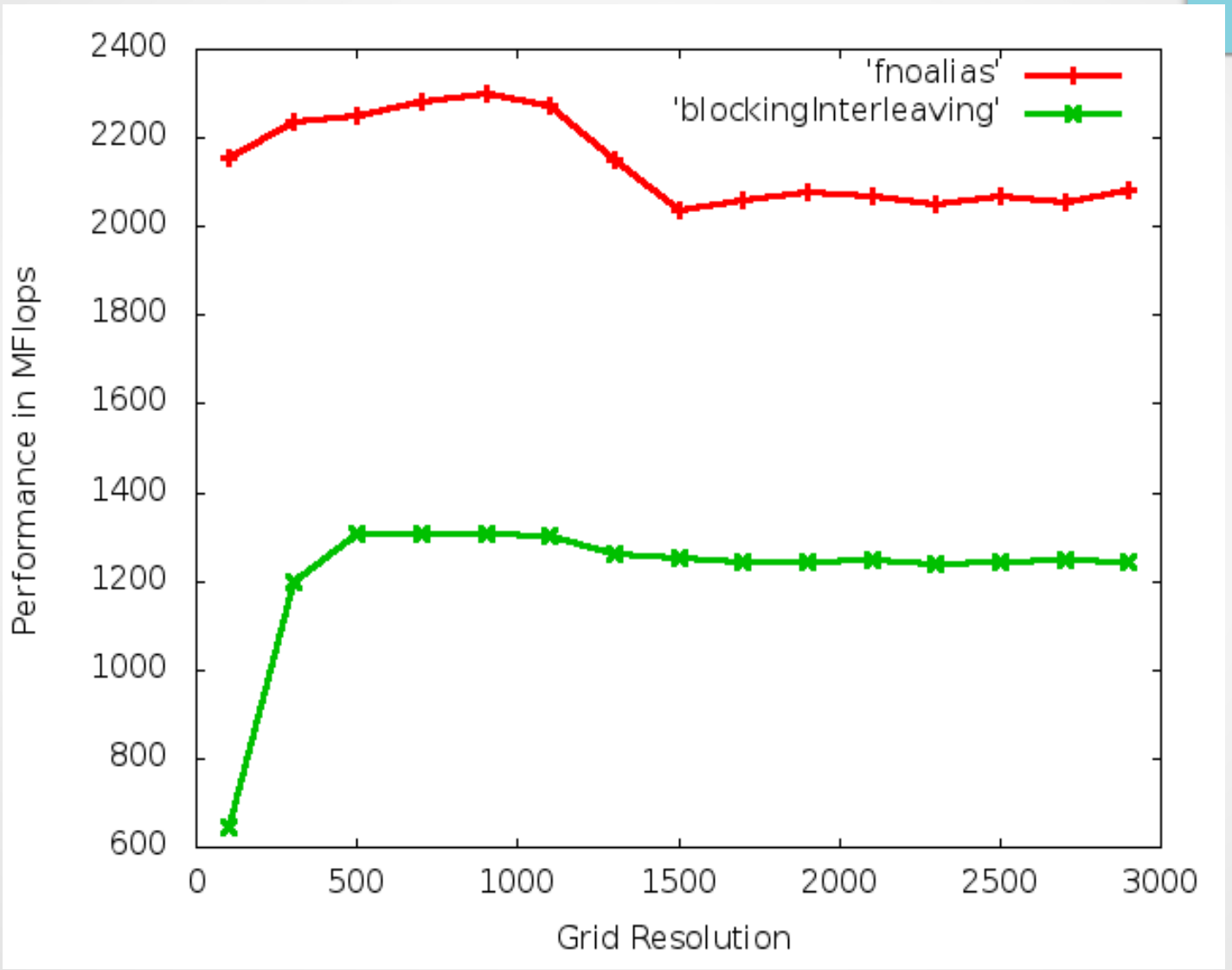
Performance Improvement



No Aliasing boost

- -fargument-noalias : option tell compiler that function arguments cannot alias each other
- It gave a huge boost, runtime down to 6 sec from 9 sec

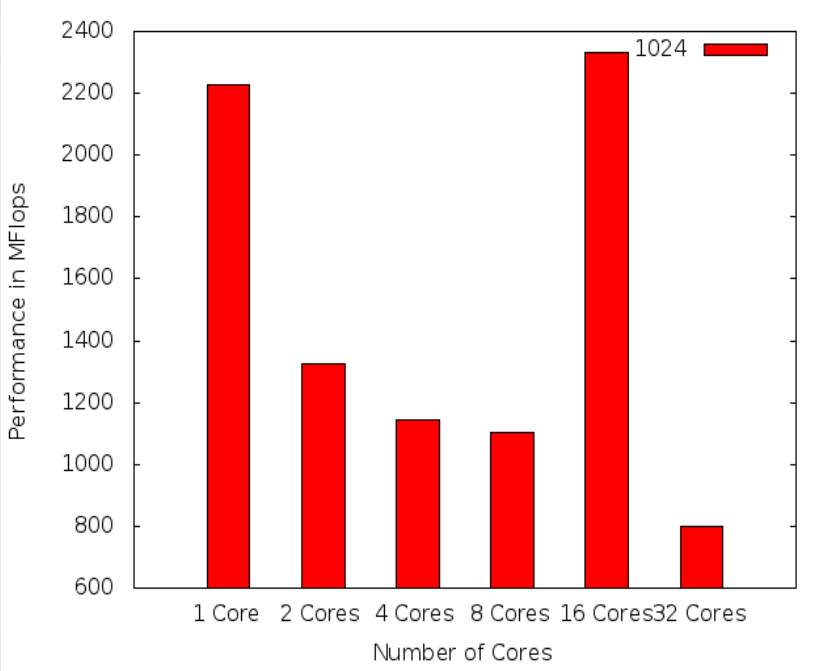
Fnoalias boost

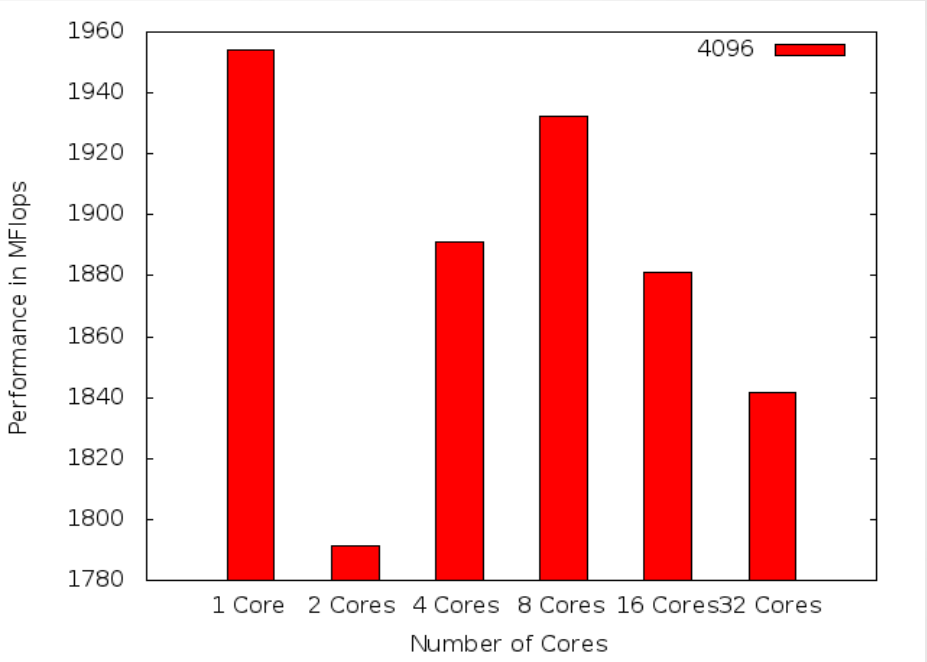


Automatic parallelization pragmas

- #pragma parallel
- #pragma loop count min(*)
- #pragma ivdep
- #pragma parallel always

Automatic parallelization results





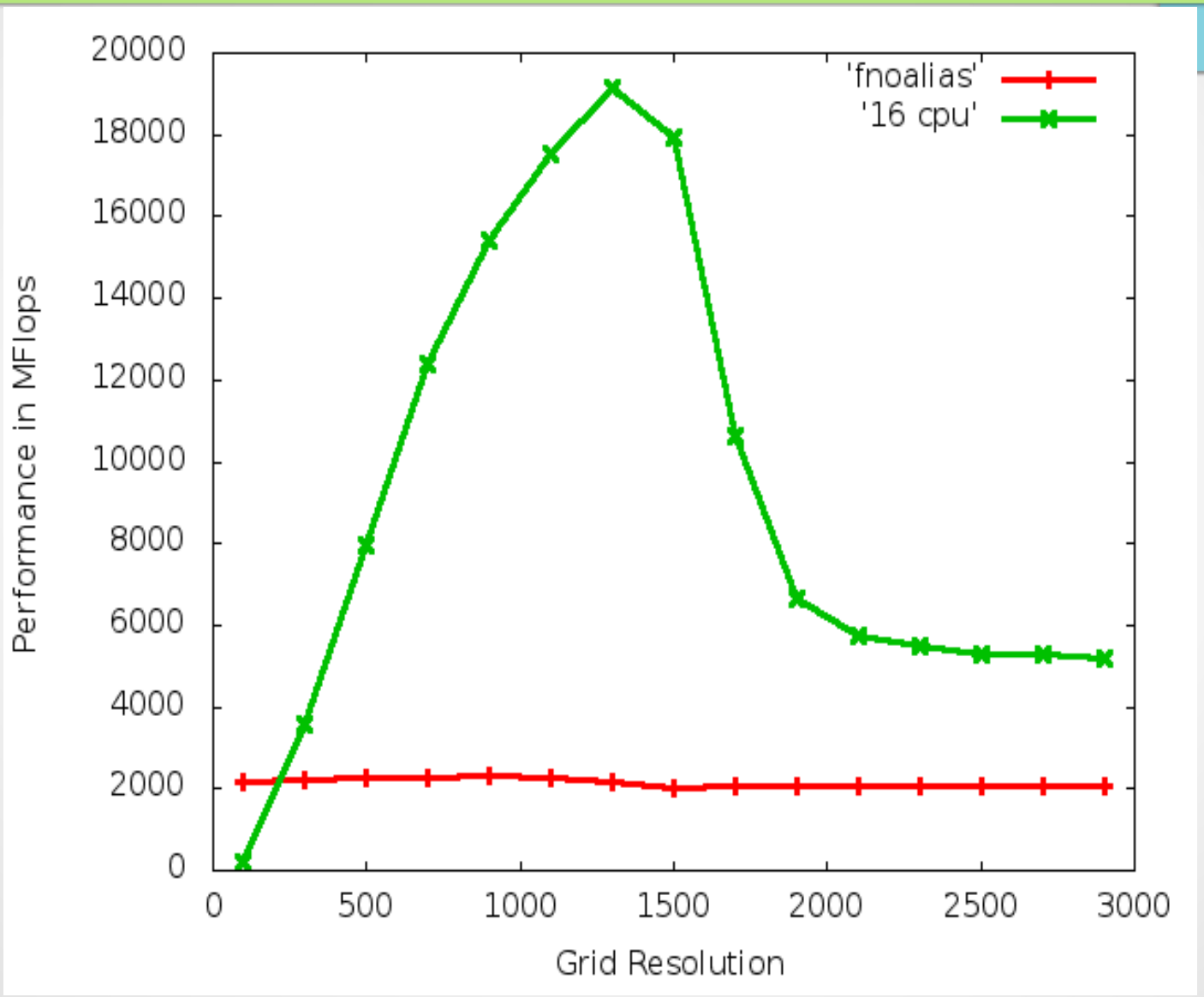
Why Automatic parallelization failed?

- Used -par-report3 to generate autoparallelization report
- Vector dependencies, tried to remove with #pragma ivdep
- “Loop not parallelized:insufficient computational work”
- Removed with “-par-threshold[n]”, n={0..100}
- “Loop not parallelized:insufficient inner loop”
- Removed with #pragma parallel always
- These workarounds resulted in overhead with autoparallelization =>degrades performance

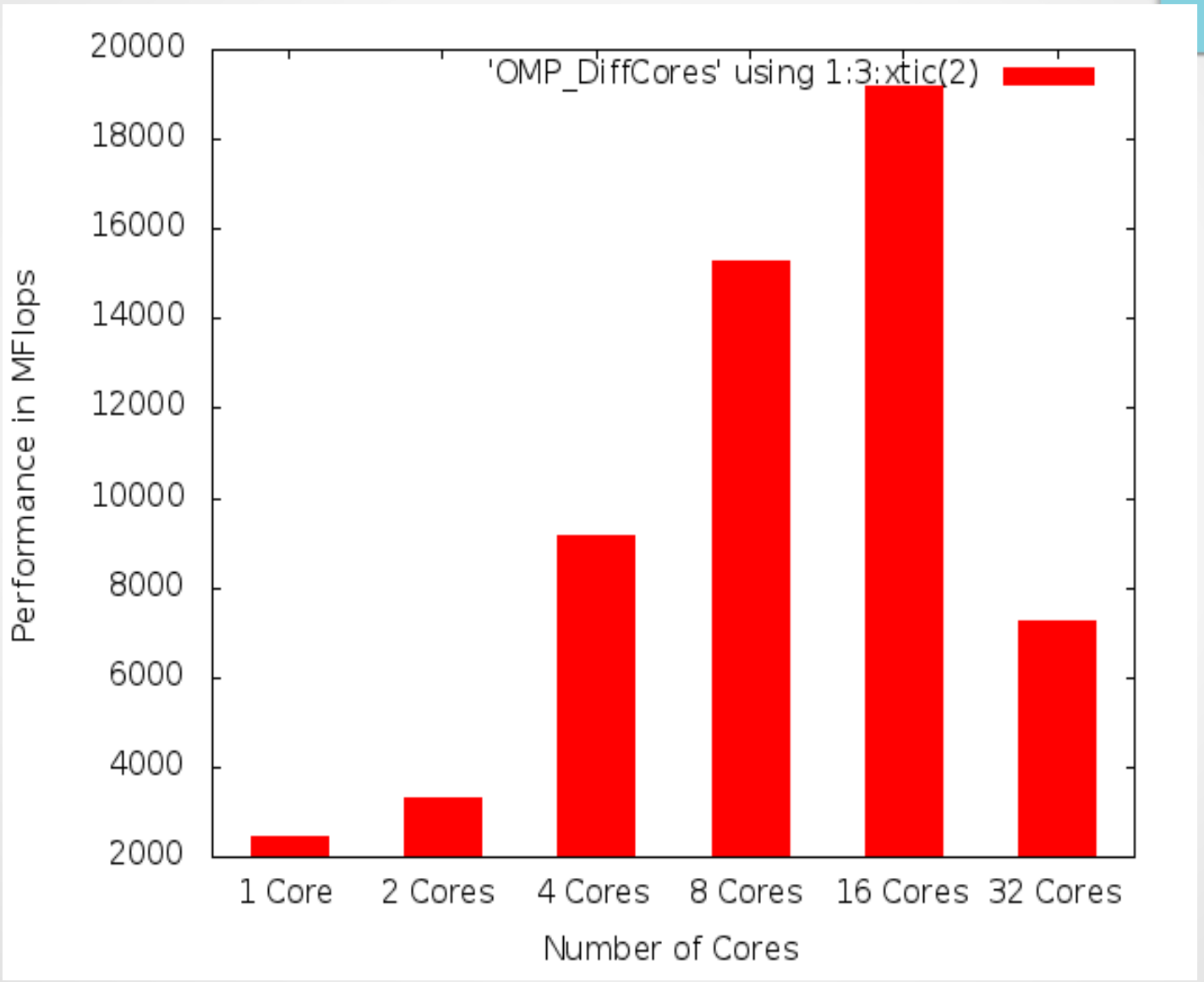
OpenMP Manual Parallelization

- omp_set_num_threads(num)
- #pragma omp parallel
- #pragma omp for private(diff) reduction(+:sum)

OpenMP Boost



OMP performance on different #cores



NUMA first touch allocation

- Modern operating systems all use virtual memory
- The OS typically optimizes memory allocations
  - malloc() does not allocate the memory directly
  - Only the memory management “knows” about the memory allocation, but no memory pages are made available
  - At the first memory access, the OS physically allocates the corresponding page (First Touch Policy)
- On NUMA systems this might lead to performance issues in threaded or multi-process applications...

(Reference: Numa Optimization(Intel), bit.ly/13okCjW)

Changes in code

```
// initialize data
for (size_t i = 0; i < N; i++)
for (size_t j = 0; j < M; j++) {...}

// perform work
#pragma omp parallel for private(j)
for (size_t i = 0; i < N; i++)
for (size_t j = 0; j < M; j++) {...}
```

CHANGED TO

```
// initialize data
#pragma omp parallel for private(j)
for (size_t i = 0; i < N; i++)
for (size_t j = 0; j < M; j++) {...}
// perform work
#pragma omp parallel for private(j)
for (size_t i = 0; i < N; i++)
for (size_t j = 0; j < M; j++) {...}
```

Peak performance for 32 Threads increased from 7237 to 9563

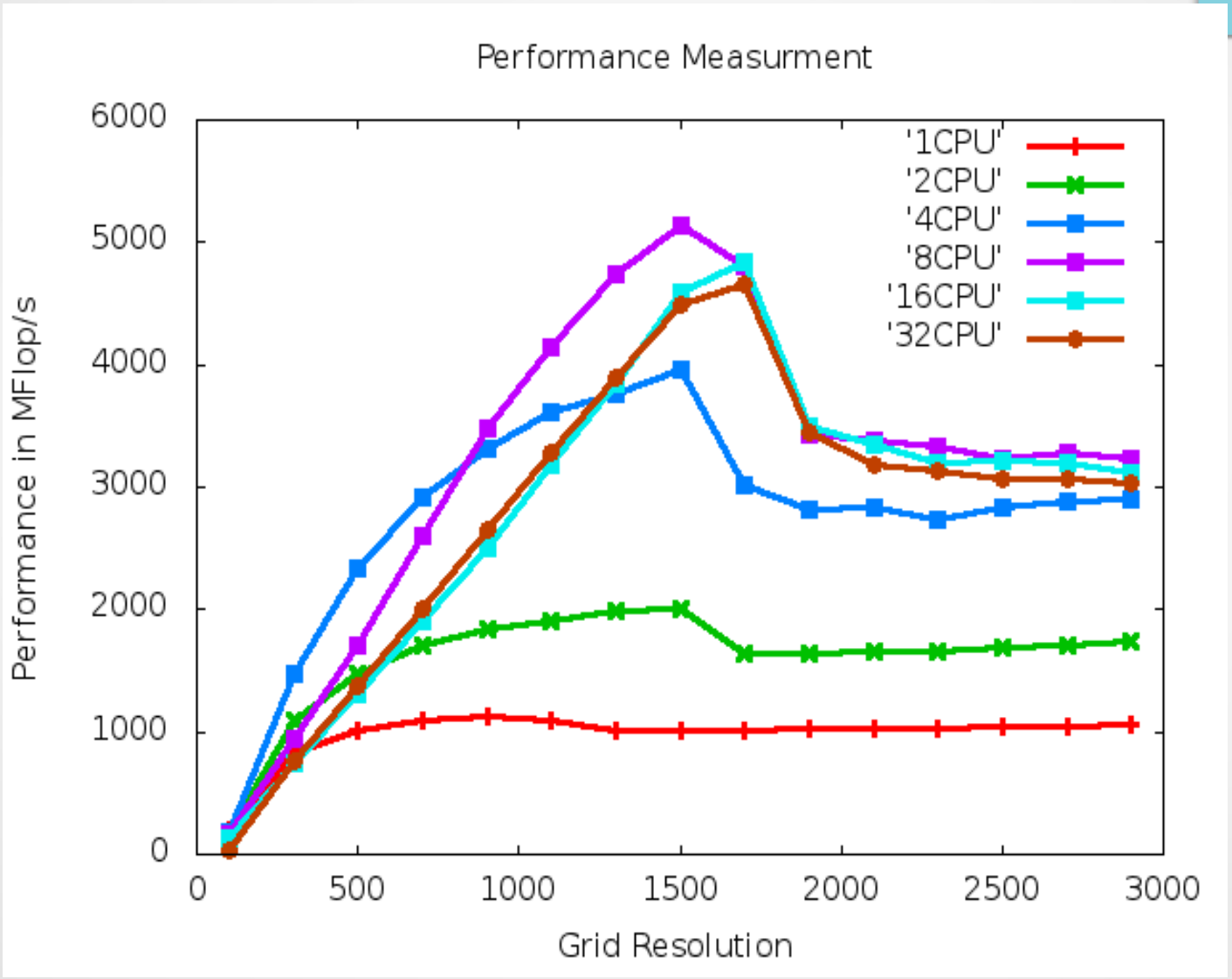
(Reference: Numa Optimization(Intel), bit.ly/13okCjW)

New Readings of Auto Parallelization, NonNuma & Numa

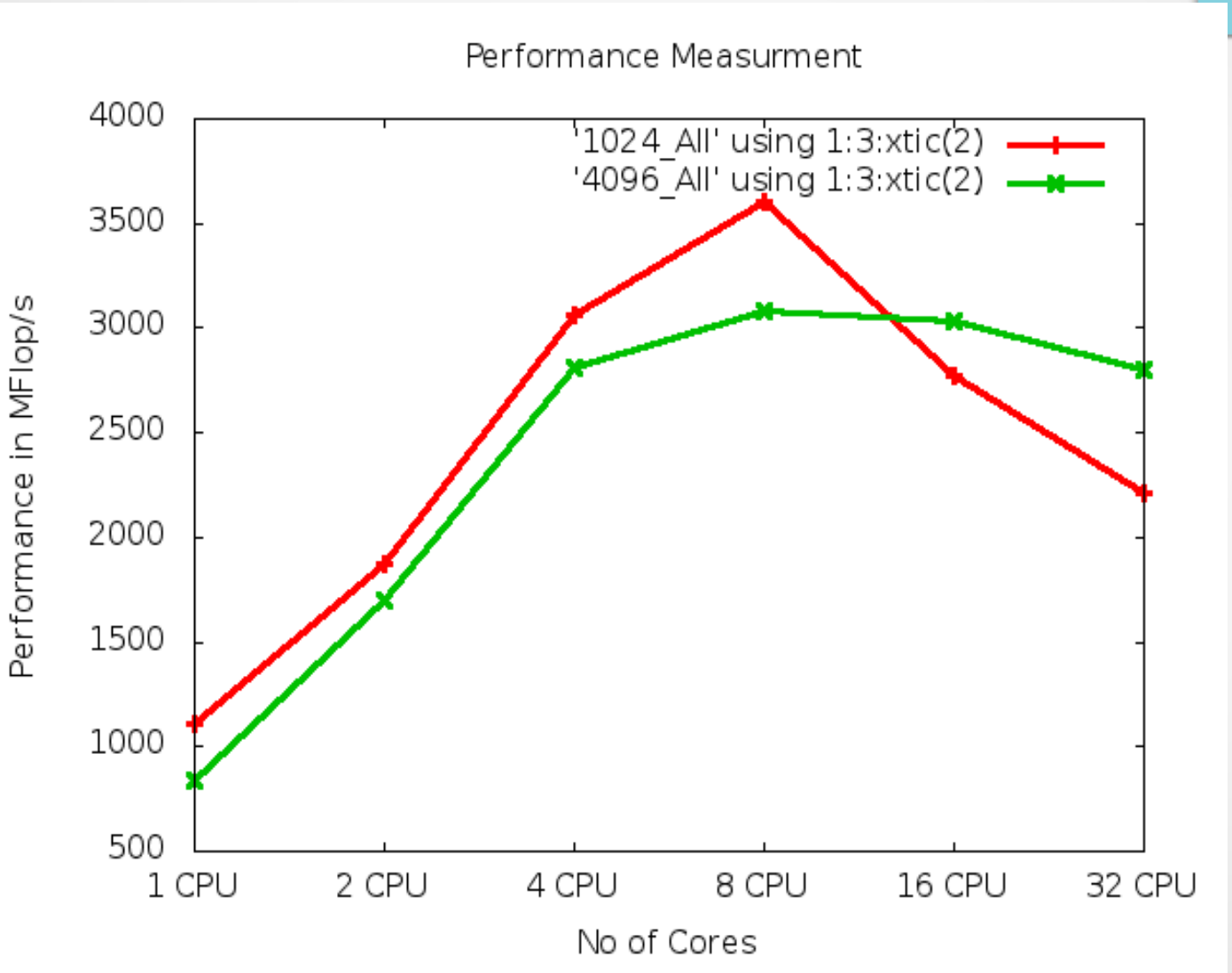
Auto Parallelization

- Auto Parallelization done on code which differs from original code with respect to following:
  - Loop interchange
  - Avoid copy operation
  - Change in the way residual is calculated

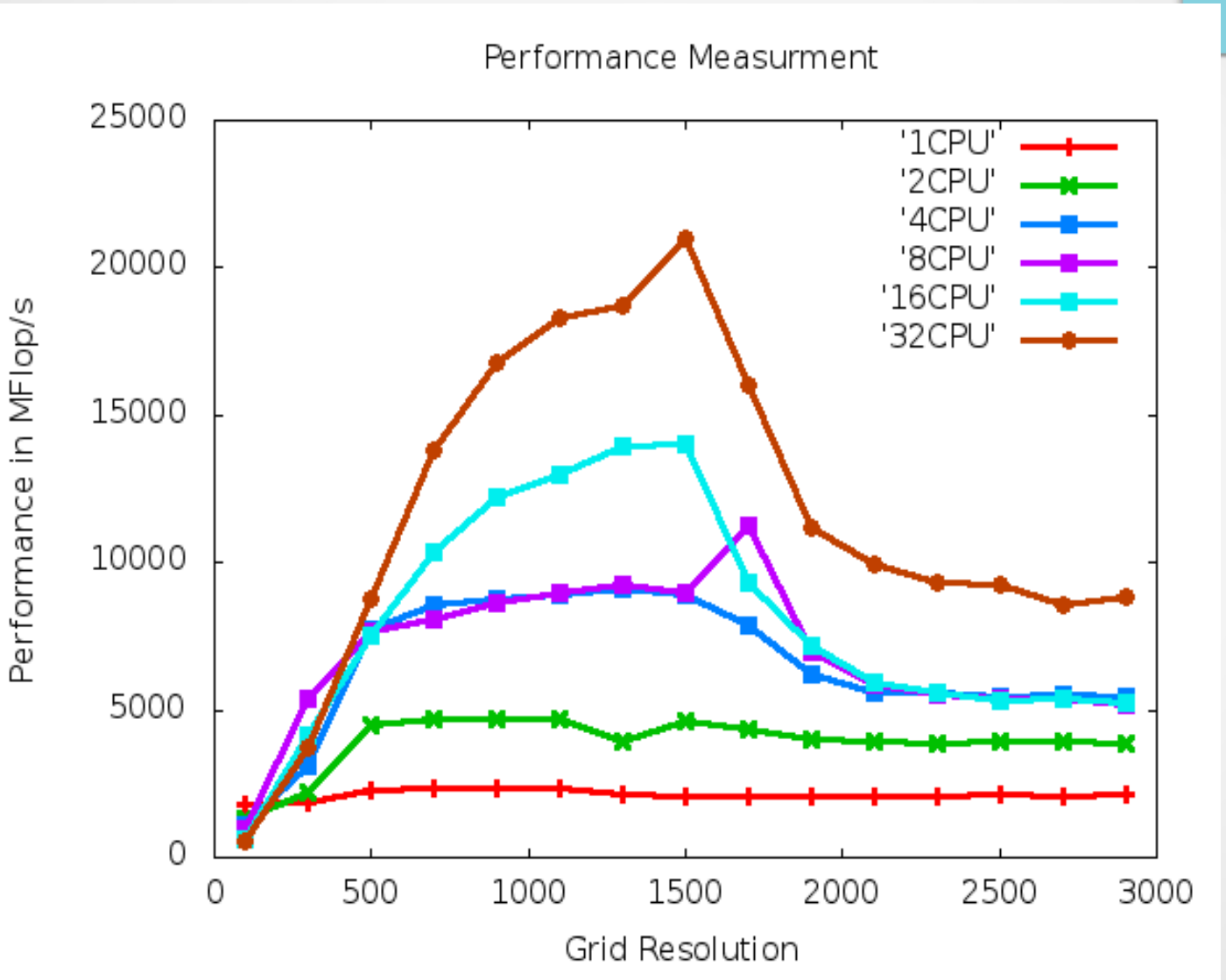
Auto Parallelization with grid resolution 100-2900



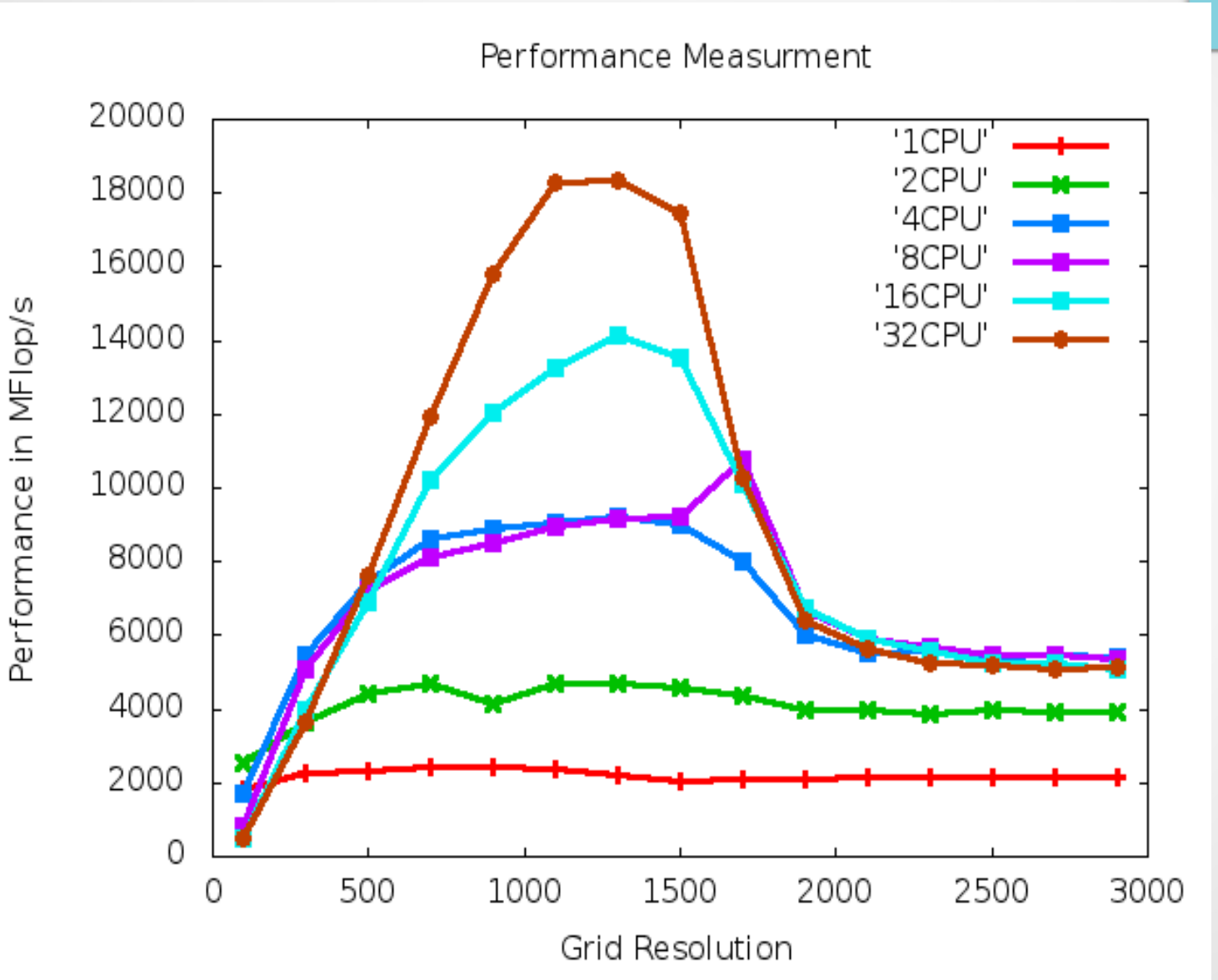
Auto Parallelize for grid 1024 & 4096



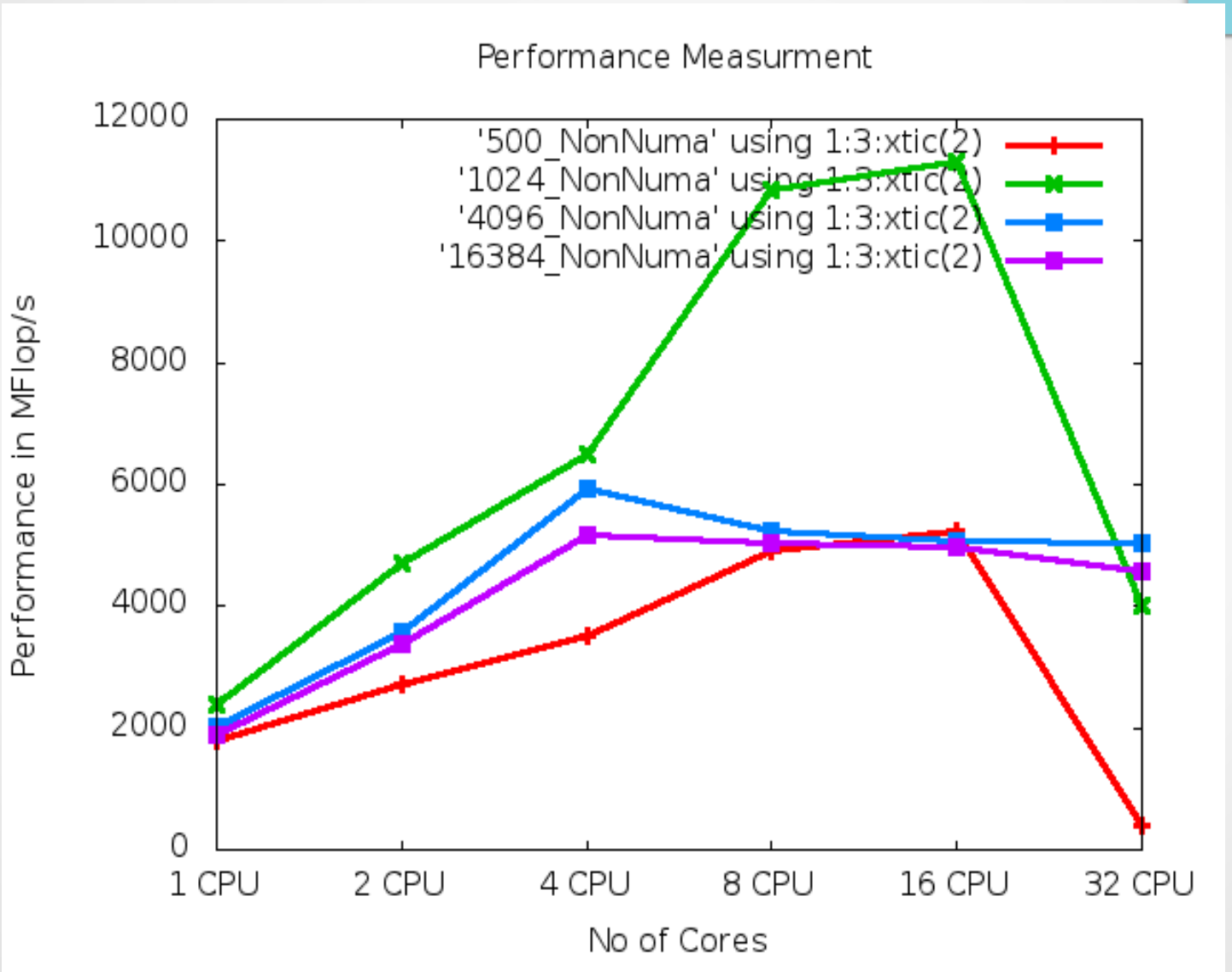
OpenMp performance without NUMA



OpenMP with NUMA



OpenMP with given grid points, non Numa



OpenMP with given grid points, with Numa