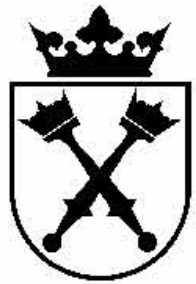
On the Importance of Memory-driven Load Balancing in Large Scale Quantum Chemical Computations

Grzegorz Mazur, Marcin Makowski and Mateusz Brela

Faculty of Chemistry, Jagiellonian University



KU KDM 2009
Zakopane

Introduction

- Chemists want more and more accurate information about bigger and bigger molecules
- Theoreticians keep providing better and better models
- Computers are getting faster and faster

Seems good. But is it really that simple?

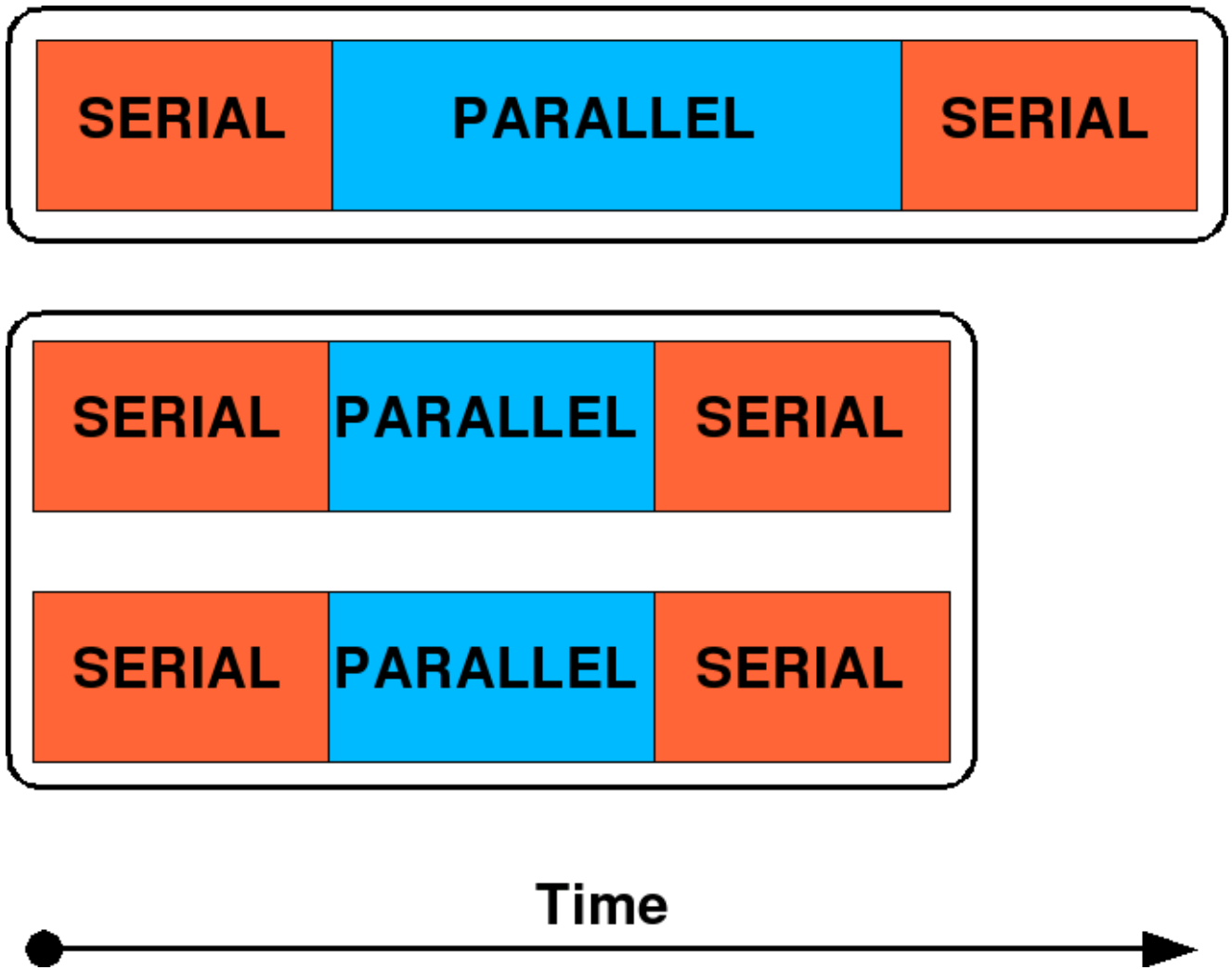
What's the problem?

- Typical computer architecture is highly parallel (supercomputers, clusters)
- Algorithmic changes are required to efficiently use available resources
- The presentation is about some of the issues we solved parallelizing our code

Parallelization made simple

Parallelization made simple

Parallelization made simple



What do we want to partition?

- Computational quantum chemistry is just a glorified name for simple tensor algebra
- The problems to be partitioned are (various types of) tensor contractions
- Among the most important are various transformations of the two-electron integrals

$$(\mu\nu|\kappa\lambda)$$

- The sheer size of the two-electron integrals tensor prevents standard algebraic treatment

Archetypical two-electron integral transformations I

- Double contraction (HF)

$$G_{\mu\nu} = \sum_{\kappa\lambda} P_{\kappa\lambda} \left[2(\mu\nu|\kappa\lambda) - (\mu\lambda|\kappa\nu) \right]$$

  - time complexity: naive approach gives $\mathcal{O}(N^4)$, quite easy to reduce to $\mathcal{O}(N^2)$, better scaling possible with some effort
  - space complexity: $\mathcal{O}(N^2)$
  - CPU bound
  - partitioning granularity: high

Archetypical two-electron integral transformations II

- Four-index transformation (MP2)

$$(ia|jb) = \sum_{\mu\nu\lambda\sigma} C_{\mu i} C_{\nu a} C_{\kappa j} C_{\lambda b} (\mu\nu|\kappa\lambda)$$

  - time complexity: naive approach gives $\mathcal{O}(N^8)$, quite easy to reduce to $\mathcal{O}(N^5)$, better scaling possible with some effort
  - space complexity: $\mathcal{O}(N^4)$, better scaling possible with some effort
  - memory bound
  - partitioning granularity: low

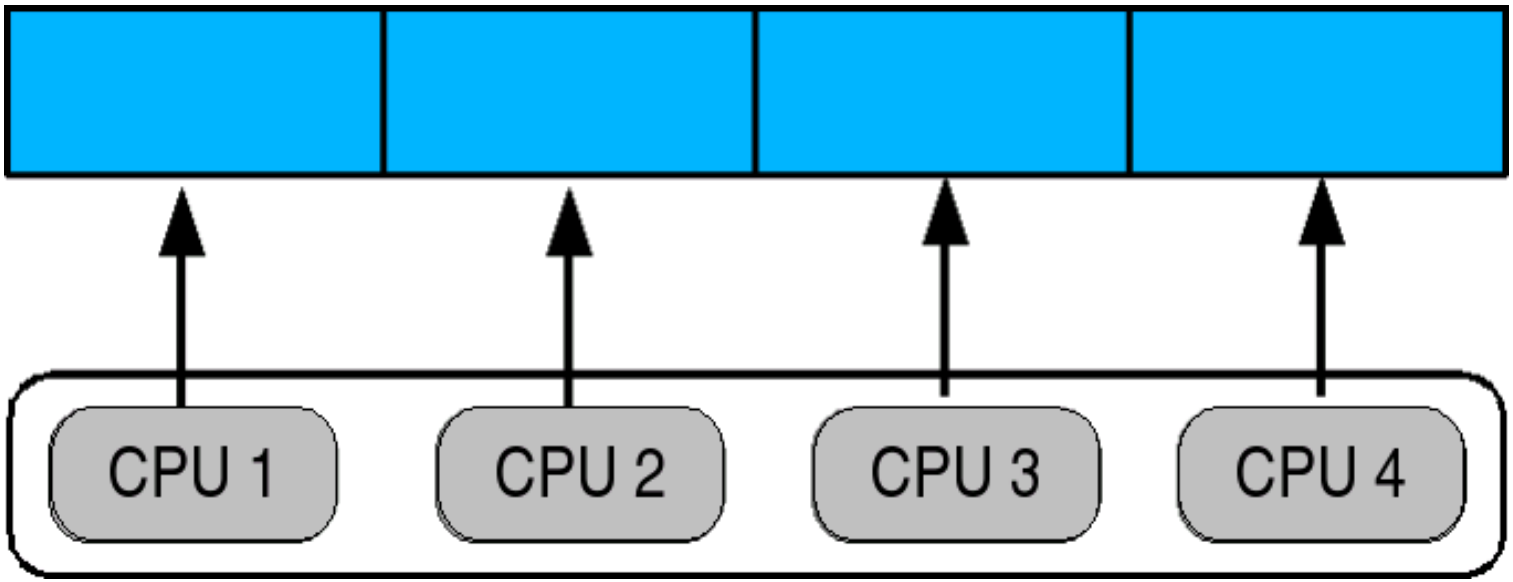
Benchmark

- To asses how well a program is parallelized, we run it several times using varying number of nodes and fit

$$T_n = \frac{\alpha}{n^\beta} + \gamma$$

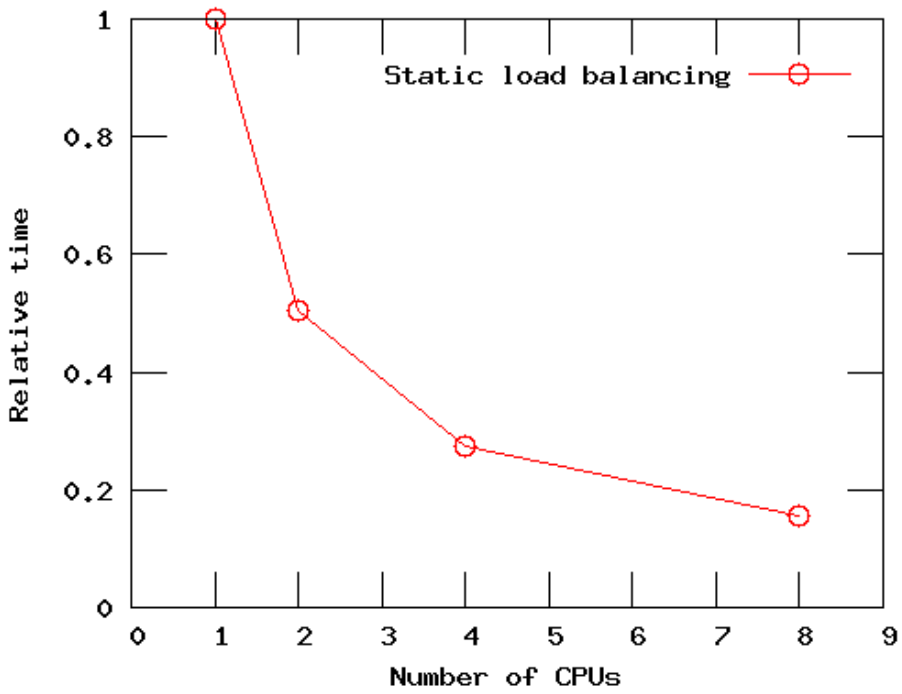
  where n is the number of nodes.
- perfect parallelization is achieved if $\beta = 1$ and $\gamma = 0$ (direct consequence of Amdahl Law)

Static load-balancing



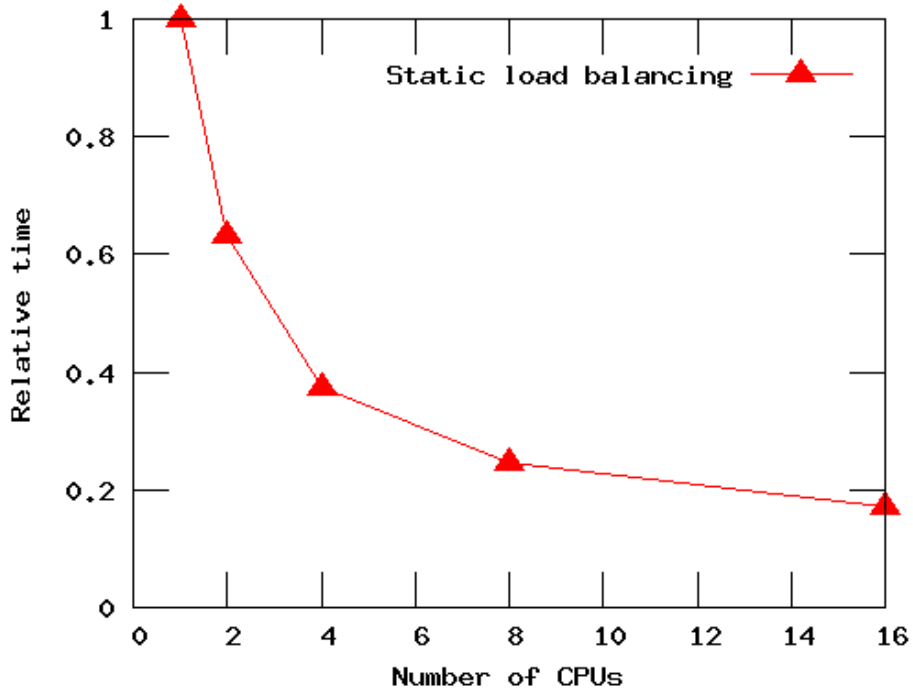
Static load-balancing results

MP2



$\alpha = 0.94 \quad \beta = 1.06 \quad \gamma = 0.05$

HF

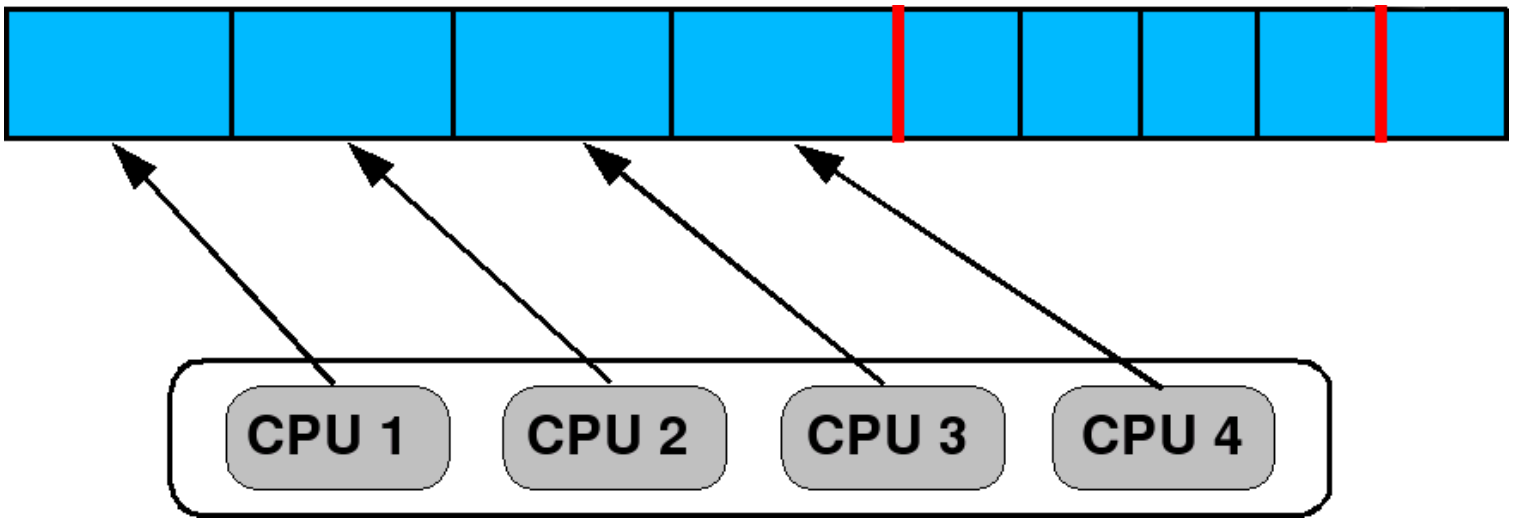


$\alpha = 0.92 \quad \beta = 0.69 \quad \gamma = 0.07$

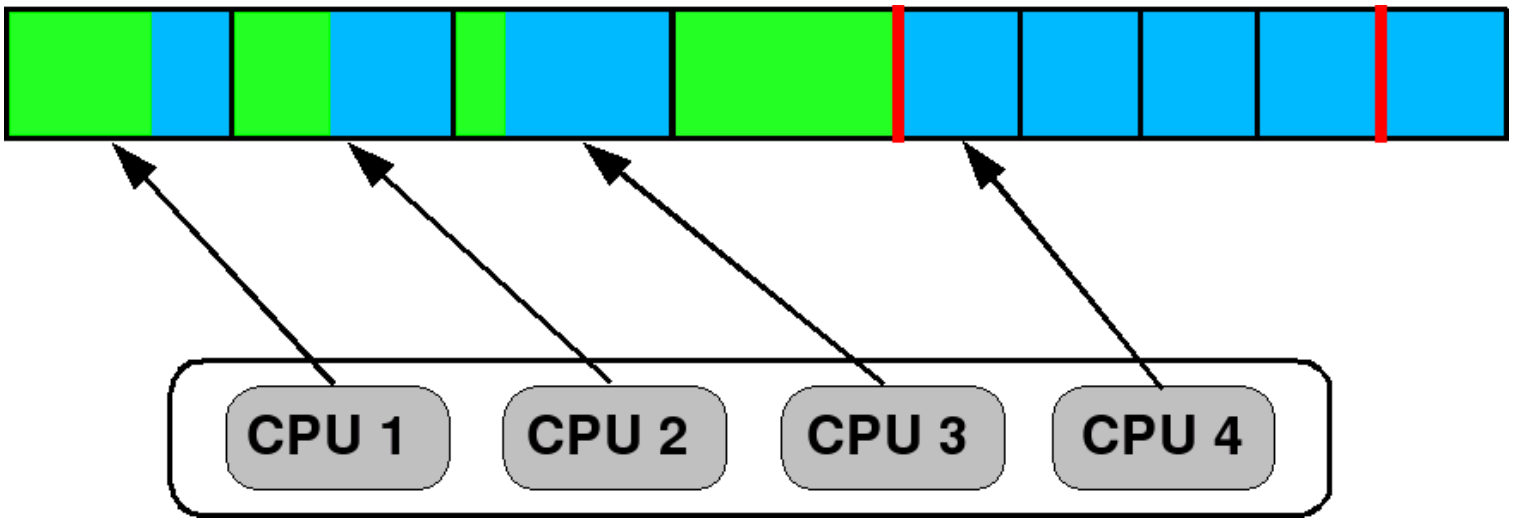
Issues with static load-balancing

- In theory, static load-balancing should be the ideal solution
- In practice, calculations on different nodes run at different pace
- The issue is inherent to large part of quantum-chemical calculations
- As a result, we have problems with synchronization

Dynamic load-balancing

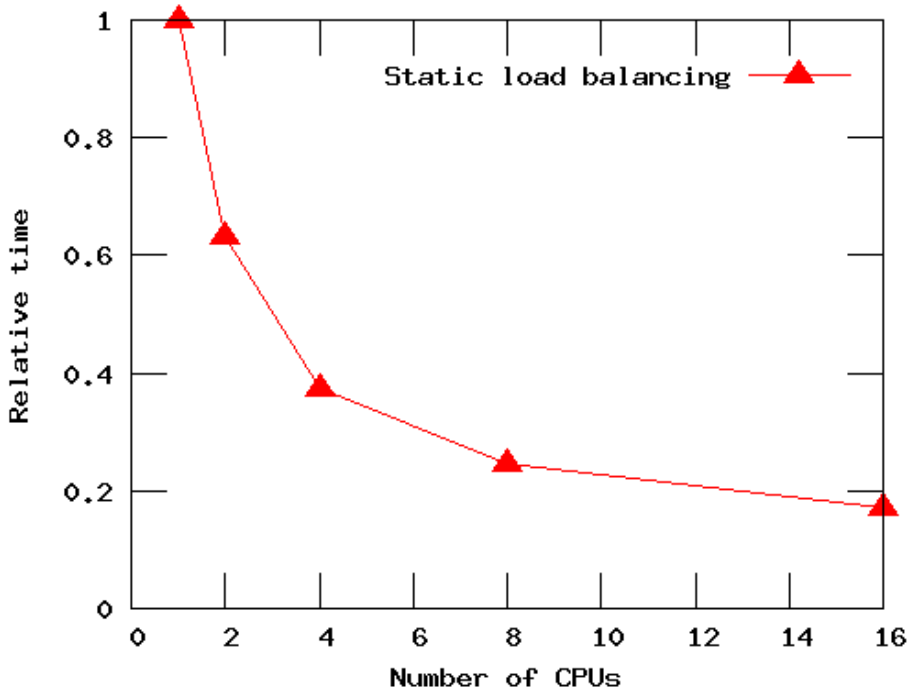


Dynamic load-balancing



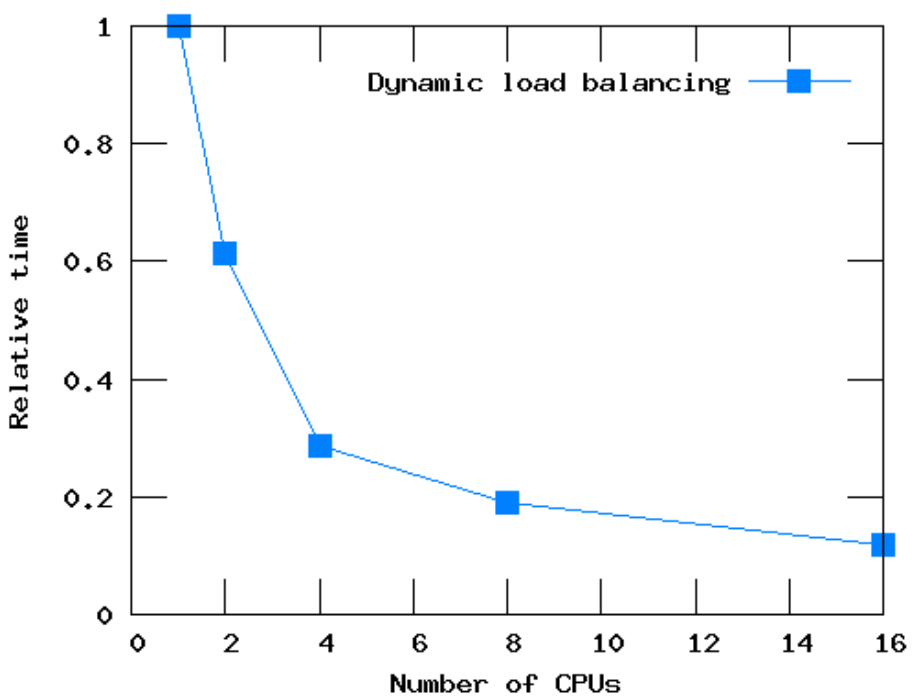
HF Dynamic load-balancing results

Static



$\alpha = 0.92 \quad \beta = 0.69 \quad \gamma = 0.07$

Dynamic



$\alpha = 0.96 \quad \beta = 1.00 \quad \gamma = 0.03$

Leave well alone?

- We are as fast as Amdahl allows, but can we do better?
- Yes, we can try to break the Amdahl limit with hyper-cache
- Hyper-what???

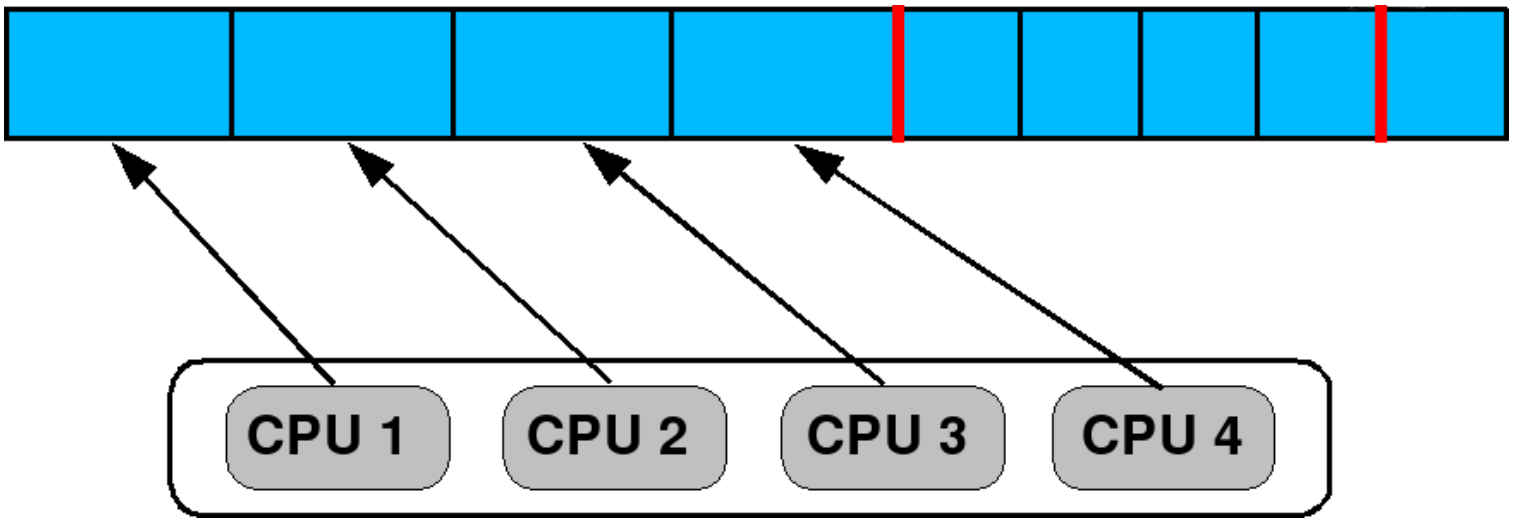
Hyper-cache effect

- In the SCF process the same values are recalculated several times
- Having more nodes we have more memory to keep the data, avoiding recalculations
- In principle this allows for breaking the Amdahl limit
- Breaking the Amdahl limit this way is called the hyper-cache effect

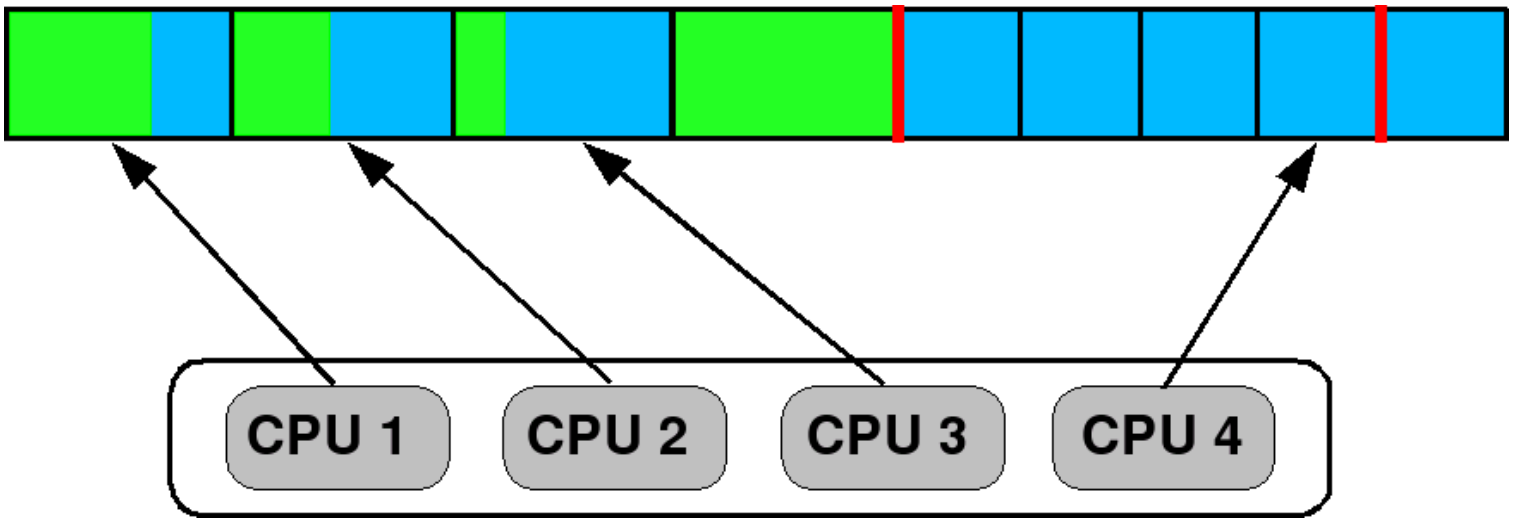
Cache-aware dynamic partitioning

- Static load-balancing maximizes cache utilization, but is very inefficient because of poor synchronization
- Dynamic load-balancing results in very good synchronization, but causes very poor cache utilization
- Is it possible to both eat cake and have it?

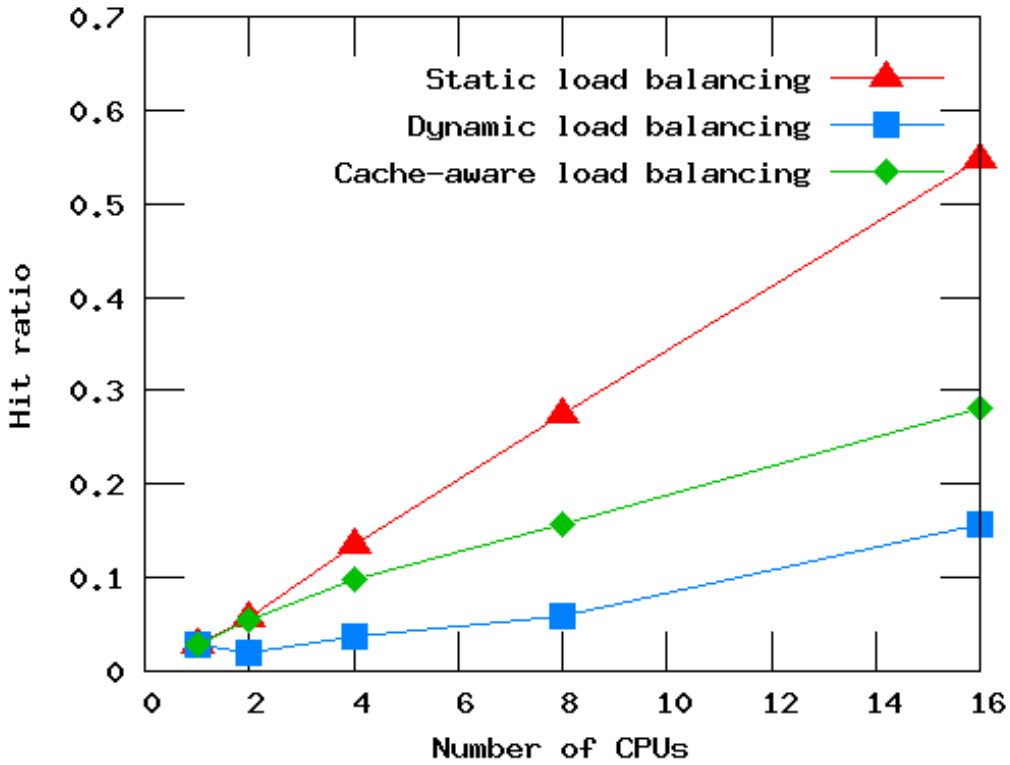
Cache-aware dynamic partitioning



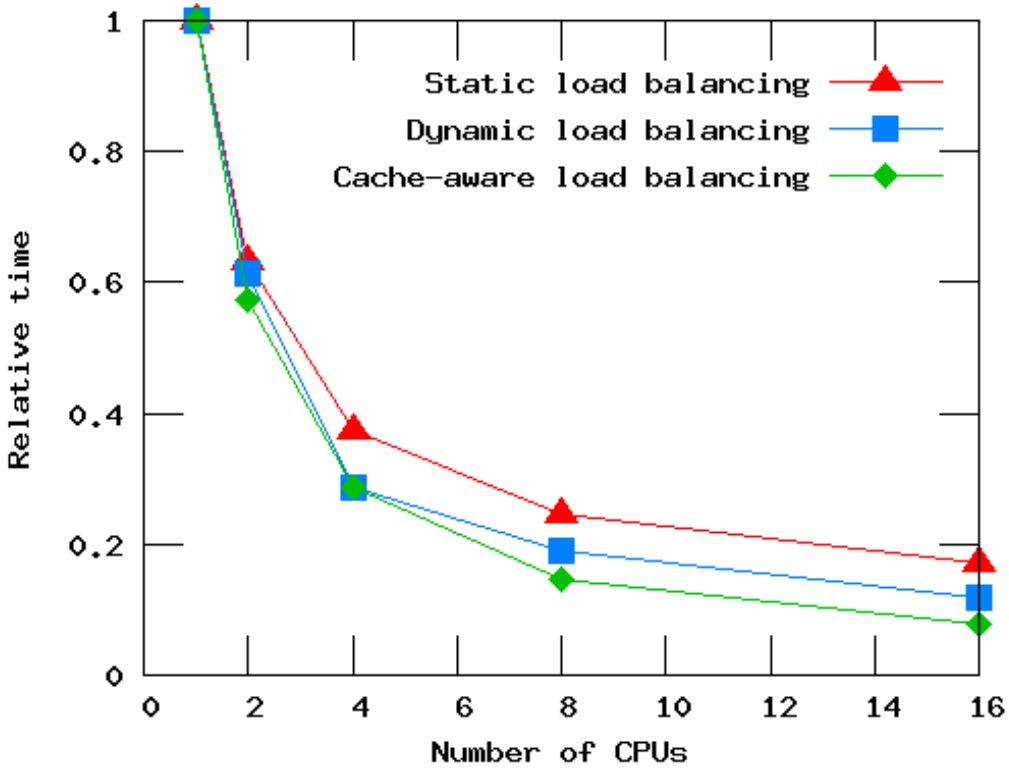
Cache-aware dynamic partitioning



HF Cache hit ratio

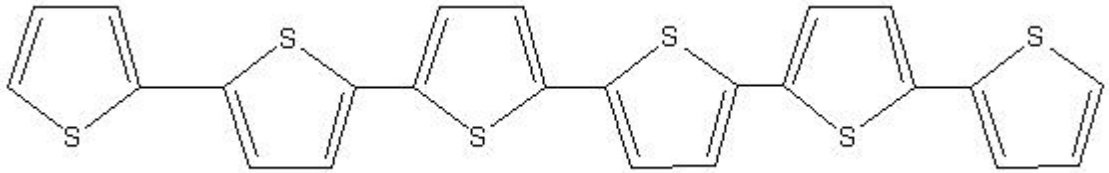


HF Results

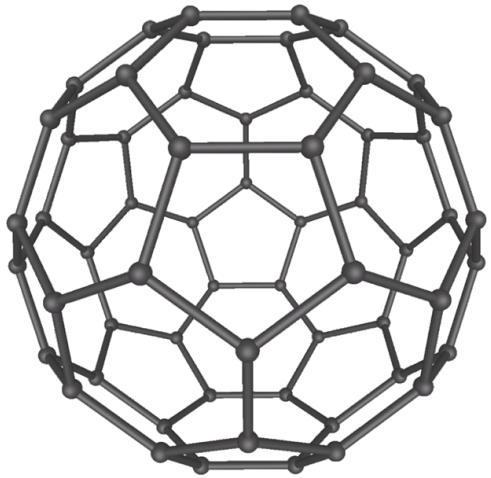


Algorithm	α	β	γ
Static	0.92	0.69	0.07
Dynamic	0.96	1.00	0.03
Cache-aware	1.00	1.07	0.02

Does it really matter?



sexithiophene: 7 min (HF/6-31G**, no symmetry)



fullerene: 1 hour (HF/6-31G, no symmetry)

Conclusions

- There is no silver bullet
  - the choice of the load-balancing algorithm depends on the job at hand
- Static load-balancing
  - fits the bill for MP2
  - performs badly for HF
- For HF calculations
  - the dynamic algorithm hits the speed limit imposed by Amdahl law
  - the cache-aware algorithm fully retains adaptivity of the dynamic one, but allows for better cache utilization
  - this results in super-linear scaling of the calculations with the number of computational nodes (hyper-cache effect)

Practical recommendations

- Running **quantum-chemical calculations in parallel makes sense**
- Owing to the number crunching/communication ratio in typical quantum-chemical calculations **clusters are the sweet spot**
- When we play the space-time tradeoff well, large per-node **memory improves performance**