



Argonne
NATIONAL
LABORATORY

... for a brighter future



U.S. Department
of Energy

UChicago ►
Argonne_{LLC}



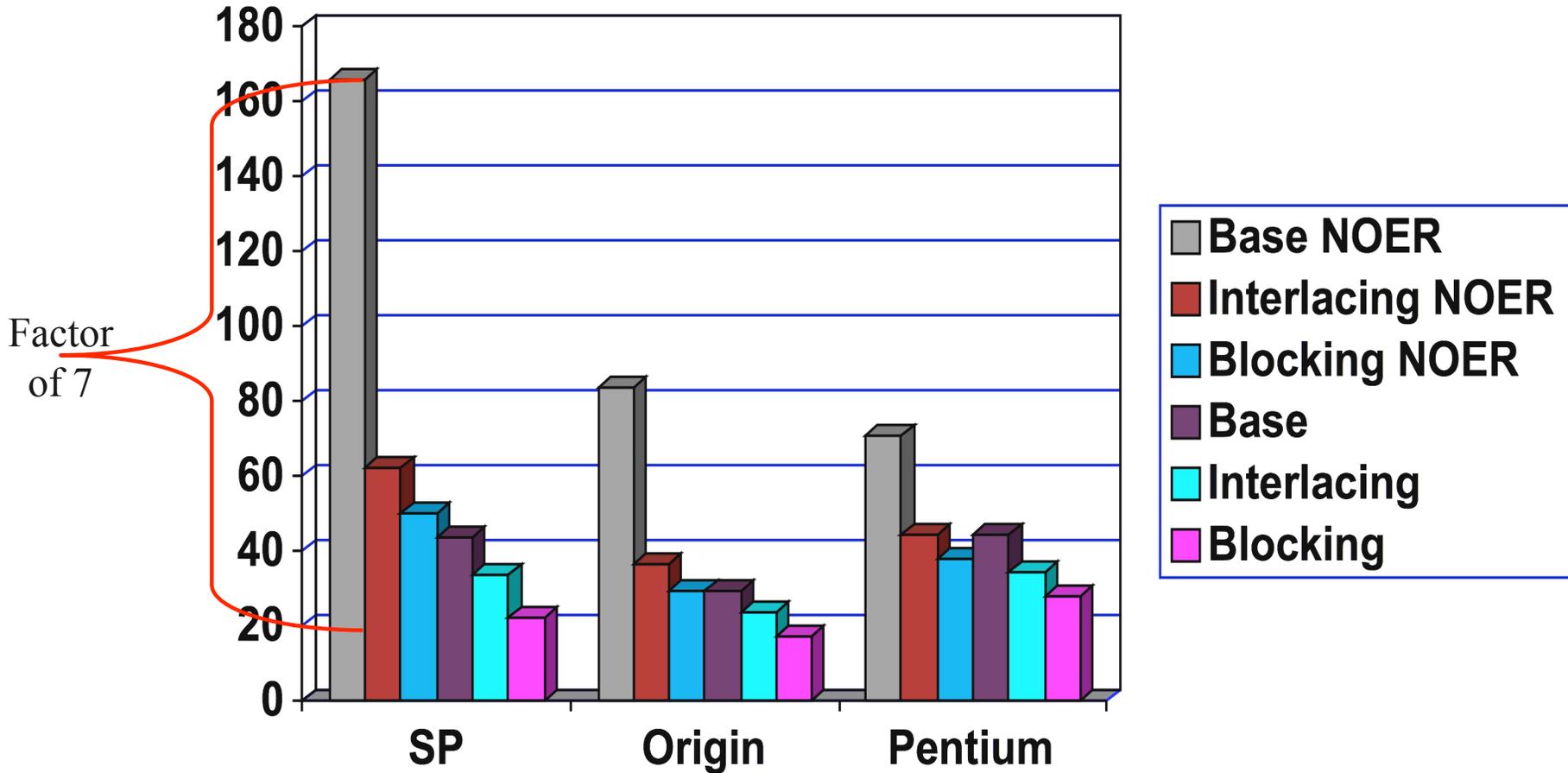
A U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC

Programming Models Summary

William Gropp

www.mcs.anl.gov/~gropp

Effect of code transformations for uni-processor performance



What Have We Done Well?

- Recognition of the importance of
 - Locality for performance
 - Portability
 - Latency hiding
 - Aiding the programmer with data decomposition/distribution
 - Reuse of Legacy code fragments

Avoiding complex memory consistency models for the programmers

– Adopting successful advances in programming models (like OO)

- Realistic global coherence model (as in, not)
- Some recognition of the problem of “brittleness”
- Some discussion of adoption strategy
- Exploring different solutions to these problems



What Are We Missing?

- Role of hardware
 - Tension between portability and performance advantages
- Need for multiple implementations of each language
- Time to develop a language
 - Are we too optimistic?
- What are the application classes?
 - We use multiple languages now
- Who is the audience?
 - Experts? Which Experts?
 - The masses?



What Are We Missing (2)?

- Intelligence and Creativity of the compiler
 - Are we expecting too much from the compiler?
 - Scalability (given problems with OpenMP, what is different)? Complexity of collective algorithms (will your compiler publish papers?)
 - Recalling the HPF experience,
 - *What features are straightforward?*
 - *What features are difficult (may involve tradeoffs wrt performance)?*
 - *What features are good ideas that require research?*
 - *How do features interact with each other, particularly WRT performance?*
- Is the real productivity problem complex software?
- Stability
 - Languages that rely on a rich set of methods are very risky (for users) because the methods are not viewed as immutable (TK, Java)
- Planning for the future
 - Will these languages be relevant when they have time to mature?
 - Heterogeneity

What Are We Really Missing?

- I/O!
- Ease of writing incorrect codes; detectibility of errors (e.g., avoiding races)
 - E.g., does deleting a token such as “atomic” introduce a race?
Would it be better to have a “nonatomic” token?
- Fault handling (defined behavior on errors, detection, repair, ...)
- Alternate (non-text-only) description formats?

