Implementing a Philosophy for Transitioning Existing Software to Network Processing

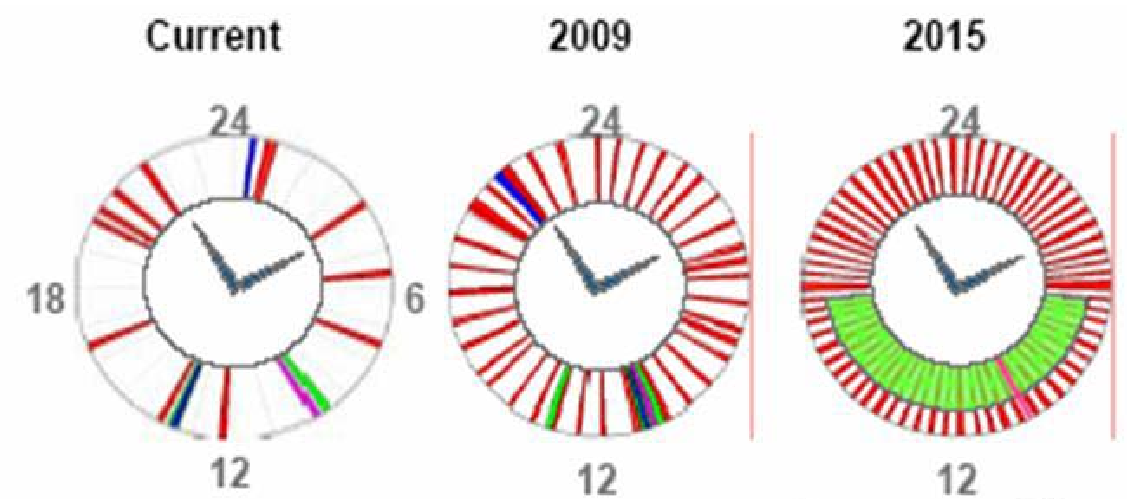
Peter Raeth
Ball Aerospace & Technologies Corp
praeth@ball.com

An Overview of our Discussion

- Roots of our customers' throughput problems
- Philosophy for addressing throughput problems
- Creating a generic extensible approach
- Applying philosophy to specific applications

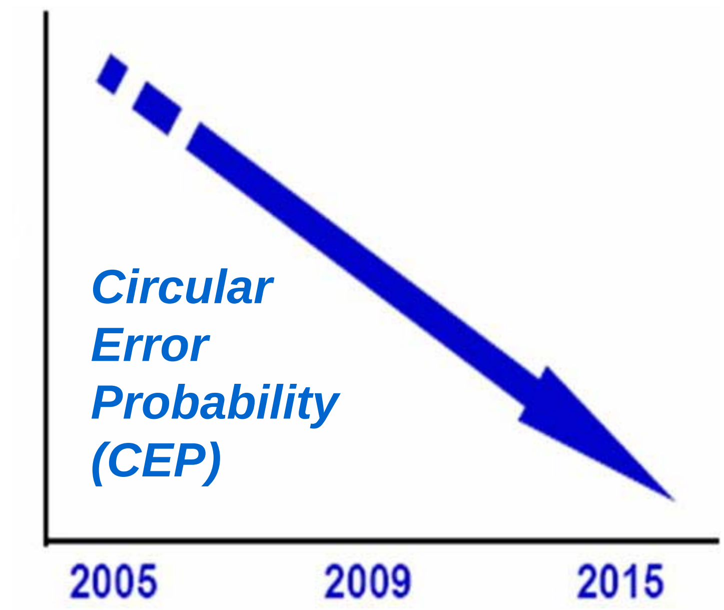
Throughput Problem Roots

Data Velocity



J. Loger, 7 Sep 2005

Calculation Accuracy



- Constant observation
- Continuous data flow
- Real-time decisions
- React to evolving environments
- Know now - Act now

- Modeling that approaches reality
- Simulation first - Construction second
- Increasing model fidelity
- Rising simulation sampling rates
- Realistic training via virtual reality

Moving Toward a General Philosophy

- Start with existing infrastructure
  - multi-processing, distributed processing, cluster processing
  - wholesale replacement is expensive, time consuming, risky
- Yet, enable opportunities in new hardware types
  - remain compatible with evolving infrastructure
  - physics chips, graphics chips, RISC, multi-core, DSP, FPGA, ASIC
  - low-latency networks
  - increased network speed
  - increased network capacity
  - faster memory bus
  - more and faster memory
  - enlarged data storage and handling
- Open source and open standards to facilitate generality and portability
  - network messaging
  - user interfaces
  - threads
  - web services
  - job management
  - network management

Moving Toward a General Philosophy

- Establish clear path from theory to production reality
  - good applications are founded on good theory
  - but, theory alone is not enough
  - can not declare problem solved until production processes are improved
  - good theoretical foundations yield technology reusability
    - if implementation is sound

Moving Toward a General Philosophy

- Take modular and generic approach
- Problem solving should span languages and operating systems
- Software should be generic, not specific to given operating system
- View components as independent cooperative objects
  - software, algorithms, data
- Transition original code while interacting with domain experts
  - start with restructuring input and output to algorithm driver
  - produce same results when compared to original
- Do not fall for application-specific hard-wiring
  - what we learn in one application should transfer easily to new applications
- Do not accept build warnings
  - nothing should be allowed to obscure issues as the project proceeds
  - "warnings" accumulate, lead to production system failure, in our experience

Moving Toward a General Philosophy

- Take into account matters beyond hardware and software
  - customer is key focus
  - some of them have very strict transition standards
  - never ask for a waiver of those standards
  - "ilities" matter
    - **modifiability**
    - **reusability**
    - **accessibility**
    - **extensibility**
    - **scalability**
    - **flexibility**
    - **system portability**
    - **workload manageability**
    - **predictability**
    - **maintainability**
    - **usability**
    - **programmability**
    - **net centricity**
    - **availability**
    - **reliability**
    - **off-shelf commodity h/w, s/w**
    - **avoid proprietary components**
    - **compatibility with existing tools**
    - **integration with exiting systems**
    - **consideration of human factors**

Employ Open Source Products

- Removes cost of licensing
- Takes advantage of community's initiatives and for-fee support
- Remain generic so that choice of tool is not a major factor
- In our case, we have made the following choices
  - MPICH2 implementation of MPI
    - cluster processing
  - Condor
    - distributed processing, workflow management, job management, resource allocation
  - pthreads
    - multi processing
  - Opticks
    - data visualization and plug-in manager
  - Axis
    - translate Java classes into web services, generate Java wrappers for web services

Most organizations have existing codes

- Lots of existing analysis algorithms and simulation models
- Written over last 20 years
- Certified as accurate and complete, in the formal sense
- Very accurate and effective in the functional sense
- Low throughput relative to advancing needs
- Need timely results, within decision/action windows of opportunity
- Desire is for improved throughput without touching algorithms
- Want transition with minimal re-engineering or new infrastructure
- Networked version of codes need to produce same results as original
  - run faster with no change in functional behavior or user view of operation

Throughput Expansion in Stages

- Single-process baseline
- Improve algorithm efficiency
- Multi-processing extension
- Cluster processing expansion for single jobs
- Distributed processing for multiple jobs
- Generic framework for porting existing codes to network environment

---

- Allows for phased improvement in consultation with domain experts
- Avoids rapid climb in complexity
- Makes best use of existing infrastructure
- Moves technical staff from what they know to what they need to learn
- Encourages coaching and teaching, marked by patience
- Organization can gradually embrace new approach to systems

Partitioning Existing Programs

- Data and Functional Partitioning
- Employ either or both, depending on application
- Look for opportunities where computation would exceed message passing in networked version

3 Basic Ways to Achieve Data Partitioning

- By data group
  - each network process gets a number of data groups to work with
  - node holding a group performs all calculations on that group
  - example: image analysis, pixel is a data group
  - results sent to managing process
- By data group component
  - each node gets a number of a data group's components
  - node holding data group's components analyzes those components
  - example: red, green, or blue component
  - results sent to process having remaining components or to managing process
- By library component
  - combined with first two
  - each node receives its data and applies a portion of the analysis to that data
  - example: material detection applies only part of spectral library to pixel subset
  - results returned to managing node or to node(s) performing other analyses

Functional Partitioning Examples

- Time-consuming modules being run in series but not dependent on outcome of preceding blocks during the same code cycle
- Pipe-line processing for cases where multiple code cycles can be going on at the same time
- Independent blocks feeding a using function
- Processes that bear no relationship to each other
- Independent processes occasionally communicating with each other

Important to Maintain Objectivity

Use clear metrics when judging throughput improvement *(S.H. Morse, p33,1994)*

- Speedup: Clock time with one process
  divided by clock time using N processes
  - ideal equals number of processes
- Efficiency: Actual speedup divided by ideal speedup
  - speedup on N processes divided by N

- Speedup and efficiency should not be confused
- With increasing speedup, it may be possible to meet throughput goal
- But, may be necessary to add “inefficient” number of processes
- End Goal: Provide results fast enough to make a positive difference within decision/action windows of opportunity

Watch Evaluation Process

- Do not begin network computing experiments with weakest machine
- Then adding more and more capable machines
- Gives false impression that experiment exceeded theoretical limits

Avoid These Routes to Subjective Results

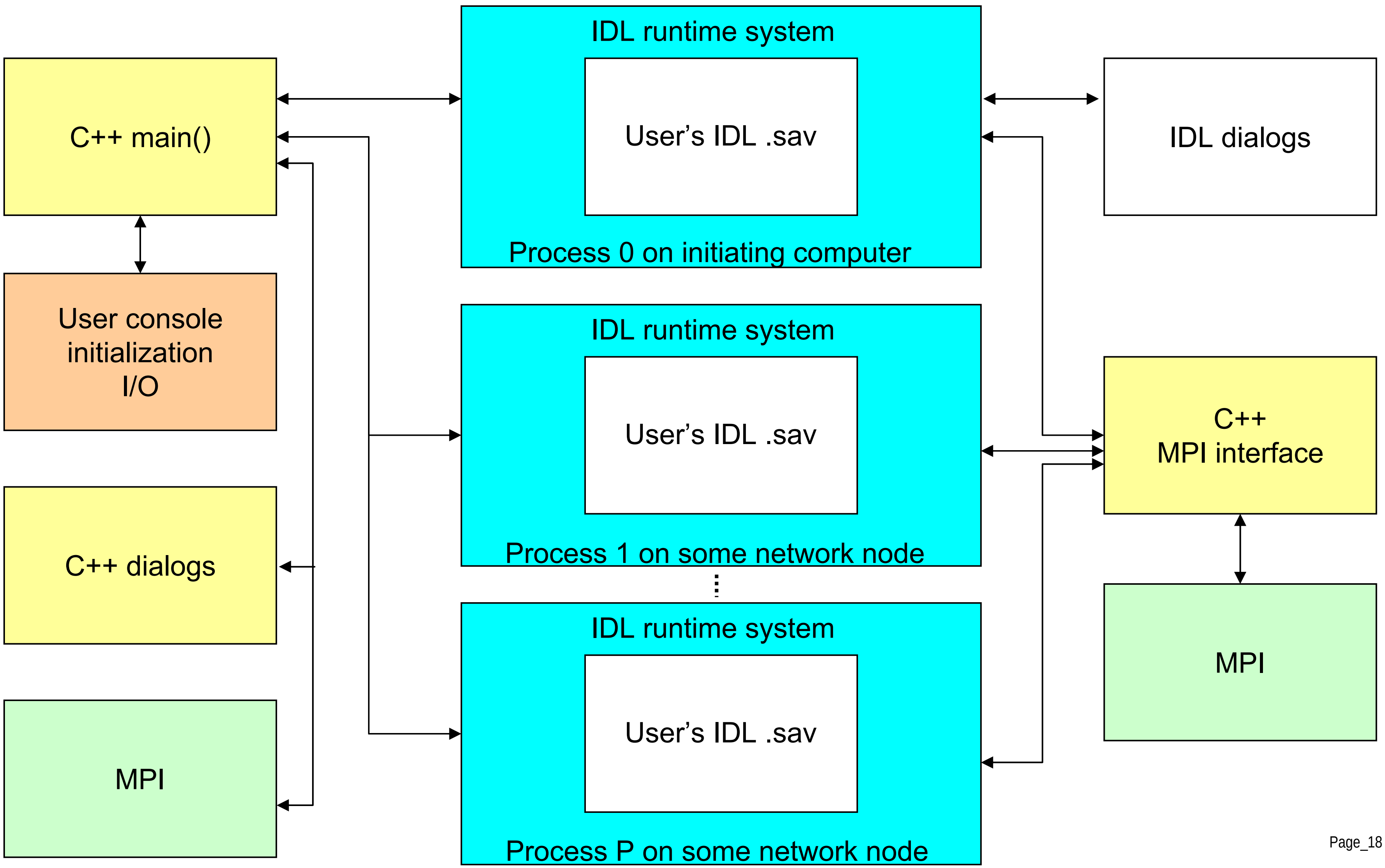
(D.H. Bailey, 1991)

- Compare 32-bit performance to 64-bit performance
- Assume inner kernel of application is sole performance determinant
- Use assembly code and other low-level language constructs for performance and compare them with Fortran or C implementations
- Scale problem size with number of processors, but do not disclose this fact
- Estimate linear scaling of performance without proof
- Compare performance of heavily optimized benchmarks against unoptimized benchmarks
- Compare with old code on obsolete system
- Base MFLOPS operation counts on the parallel implementation instead of on the best sequential implementation
- Give performance in terms of processor utilization, parallel speedup, or peak MFLOPS per $, avoiding issue of "fast enough to matter"
- Use numerically inefficient algorithms to show artificially high MFLOPS
- Measure parallel run times on dedicated system, but measure conventional run times on heavily loaded system

Success in Application

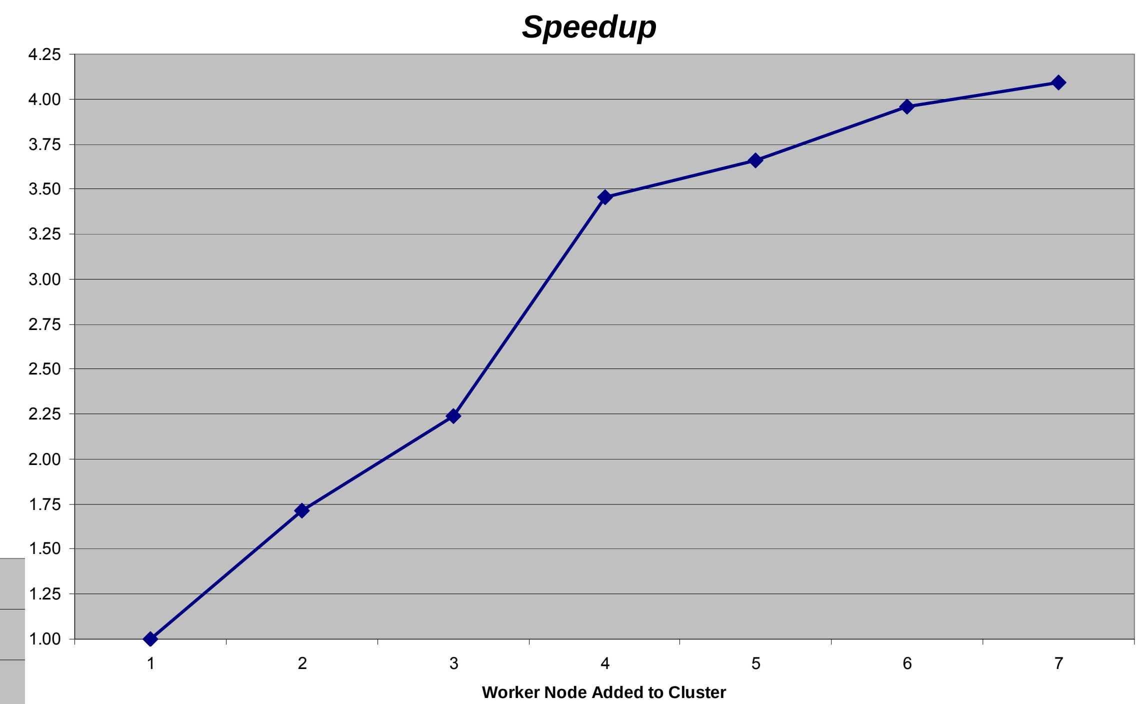
- Applied our approach to several languages and operating systems
  - Linux and Windows
  - IDL, MatLab, Simulink, C++
- Same basic philosophy applied to each
- Lessons learned on any one of these applied to all
- Enables use on plug-ins to larger tool sets
  - user interfaces
  - job management
  - workflow management
  - resource management
  - web services

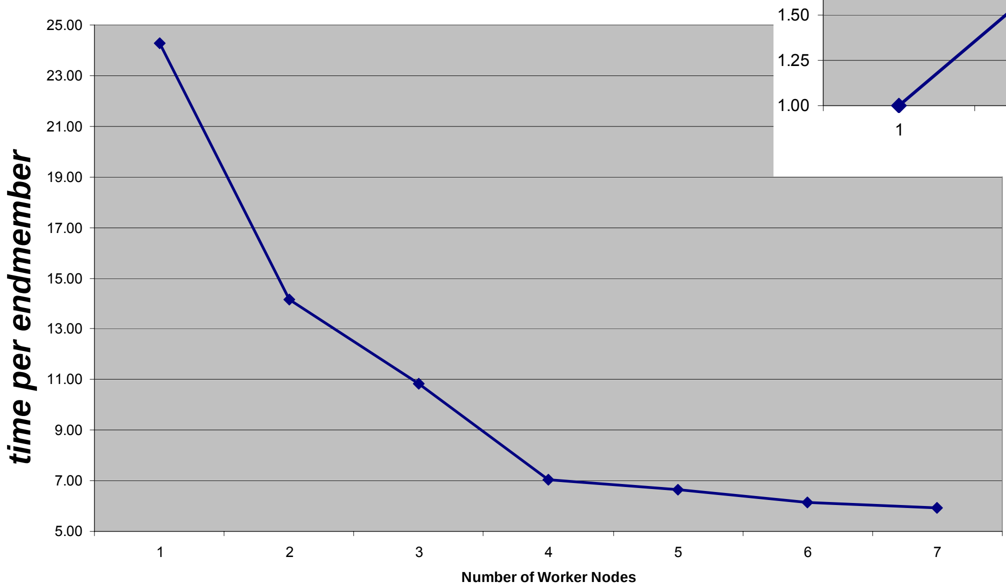
One Approach to Cluster IDL



IDL Runtime – Finding unique spectral elements

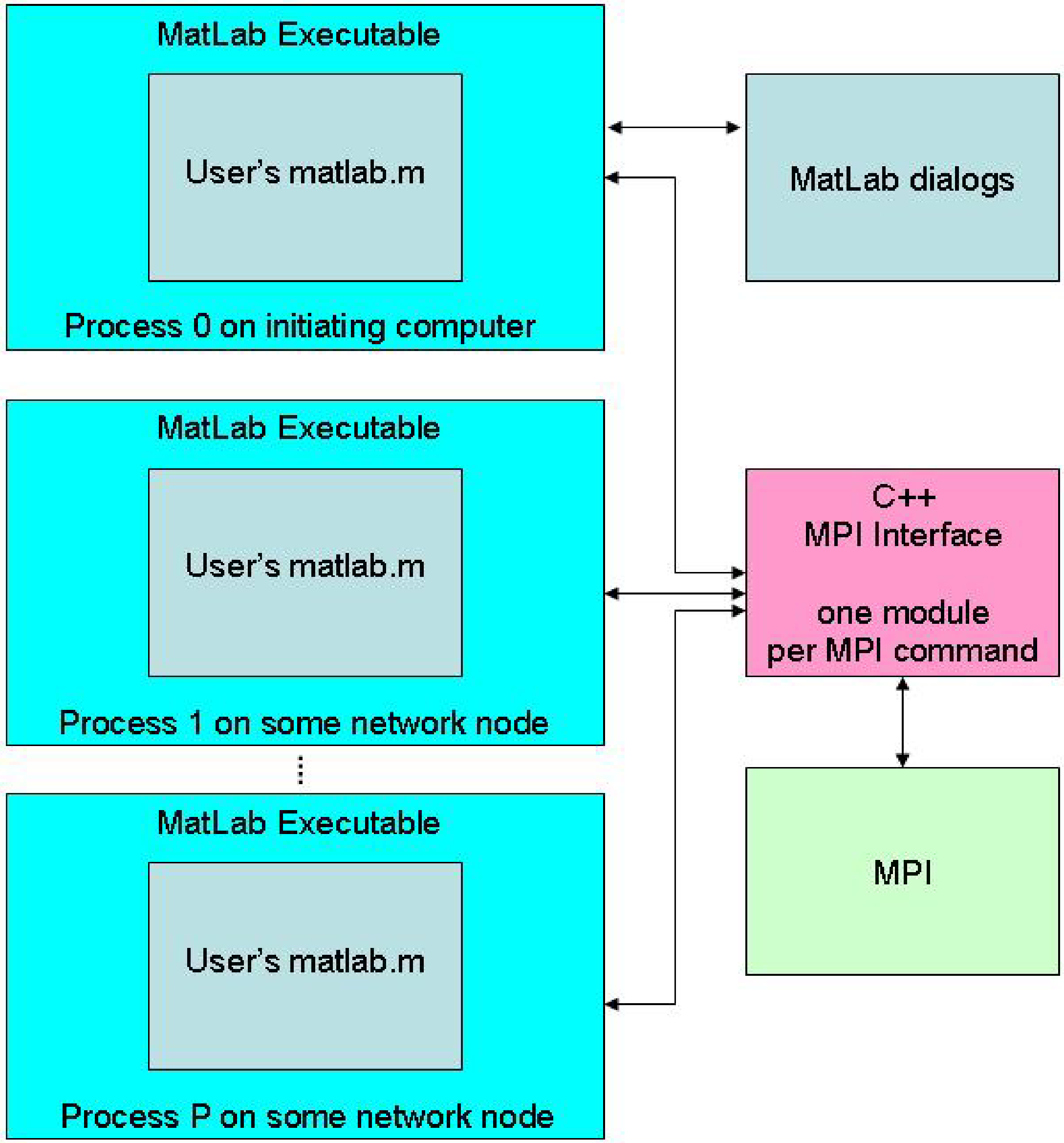
- Sought 20 unique spectral elements (endmembers)
  - hyperspectral data cube
  - 200 bands per pixel
  - 1000 x 256 pixels





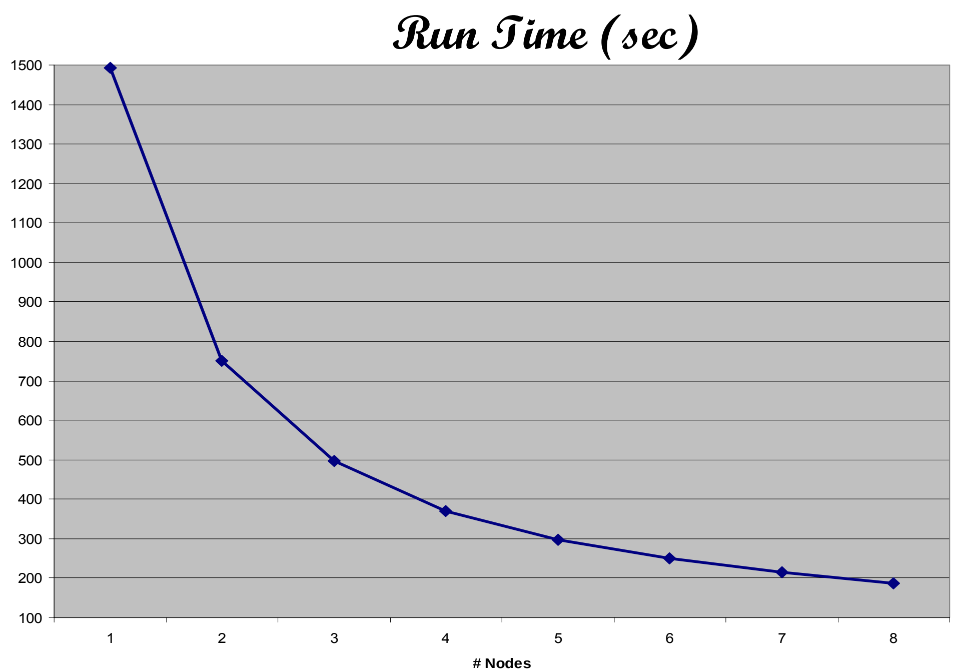
An Approach to Cluster MatLab

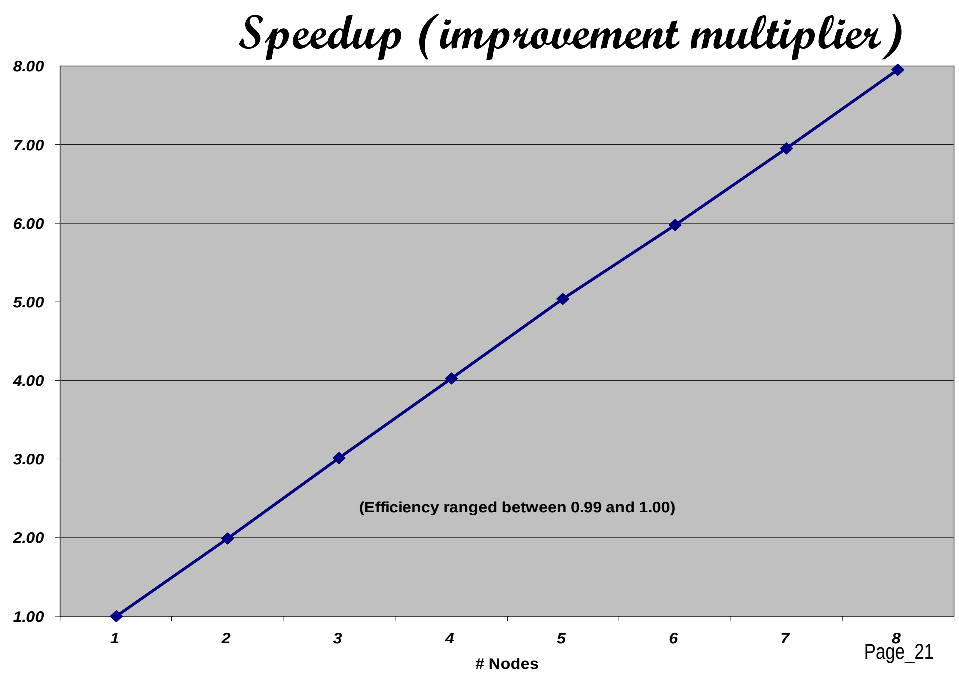
- Similar to what we do in IDL
- Avoids tool integration issues with current IDL approach
- Accommodates MatLab's "pass by value" approach to sending variables to external modules
  - MatLab paradigm based on Fortran ("pass by value")
  - MPI assumes "pass by reference"
  - MatLab can not pass new values back through function arguments



Example MatLab Run: Image Convolution

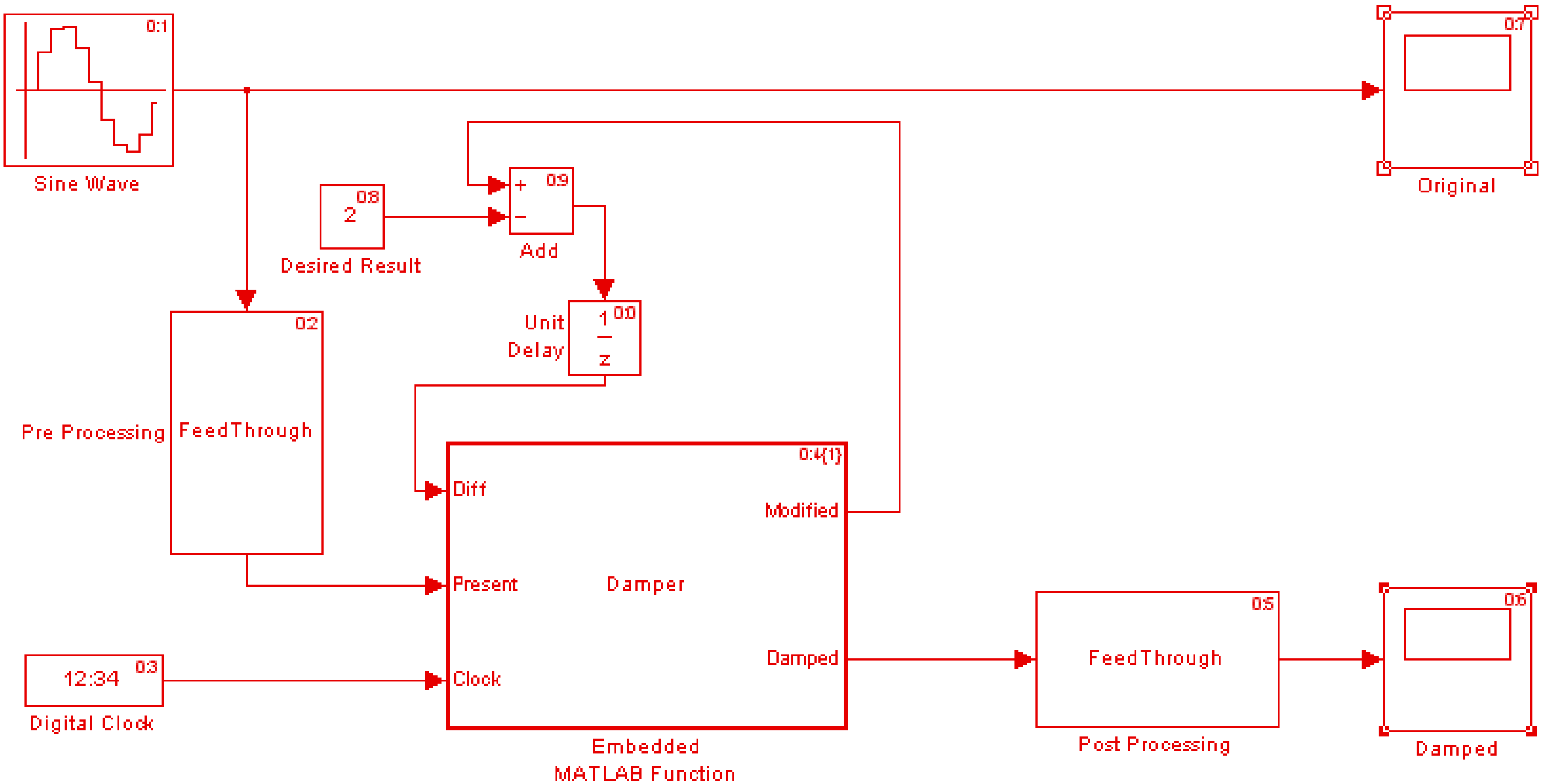
- Fundamental to image analysis
  - assigns new value to a pixel based on weighted average of neighbors
  - used to filter images in many ways for many reasons
- Convolution code written in MatLab
- 3 x 3 filter over a 2112 x 2816 grayscale image
- For this case, efficiency was nearly perfect (0.99 – 1.00)





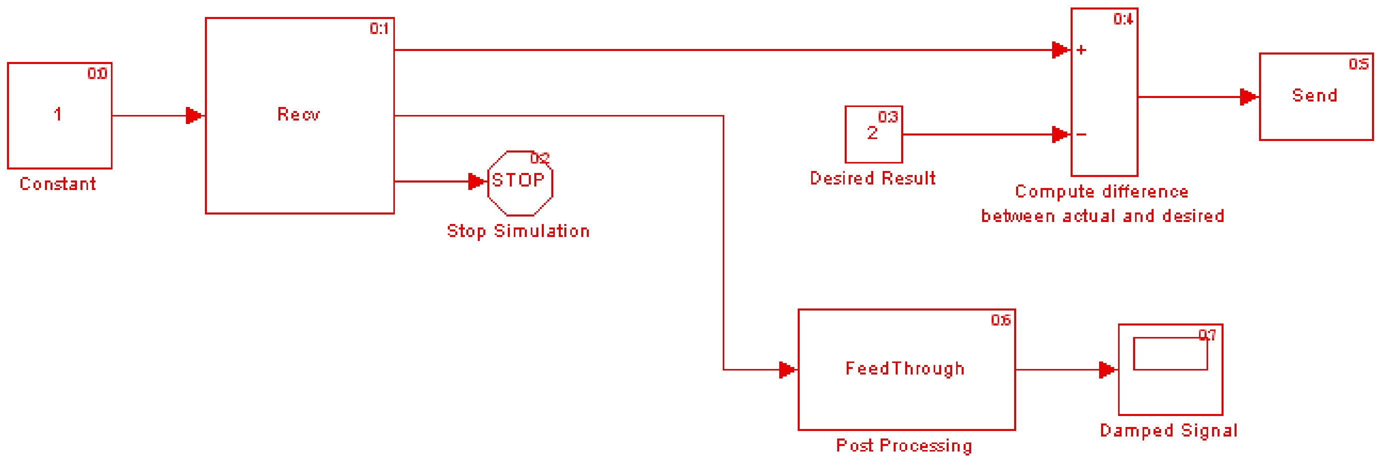
One Approach to Simulink/RTW Models

- Decompose model into independent cooperating components
- For example: dampen a signal to desired value

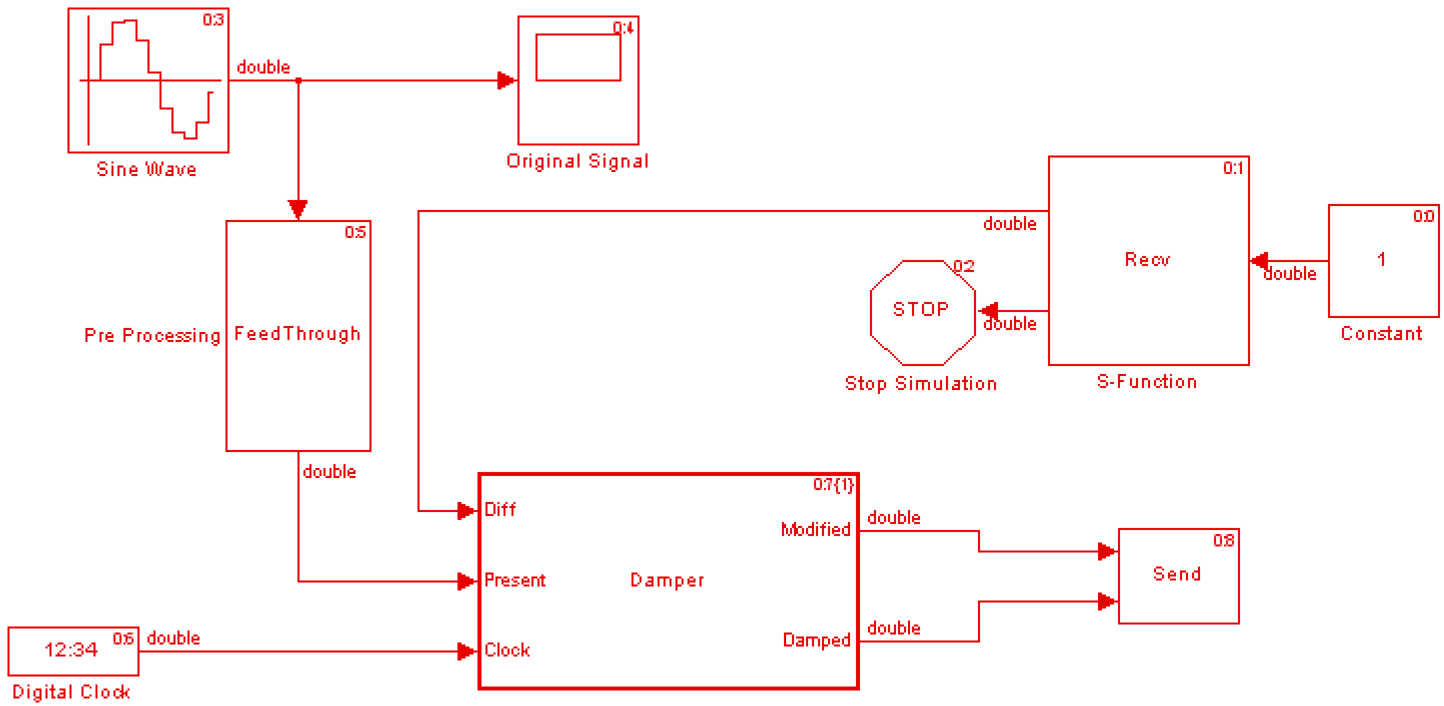


Decomposed Model
(two independent cooperating components)

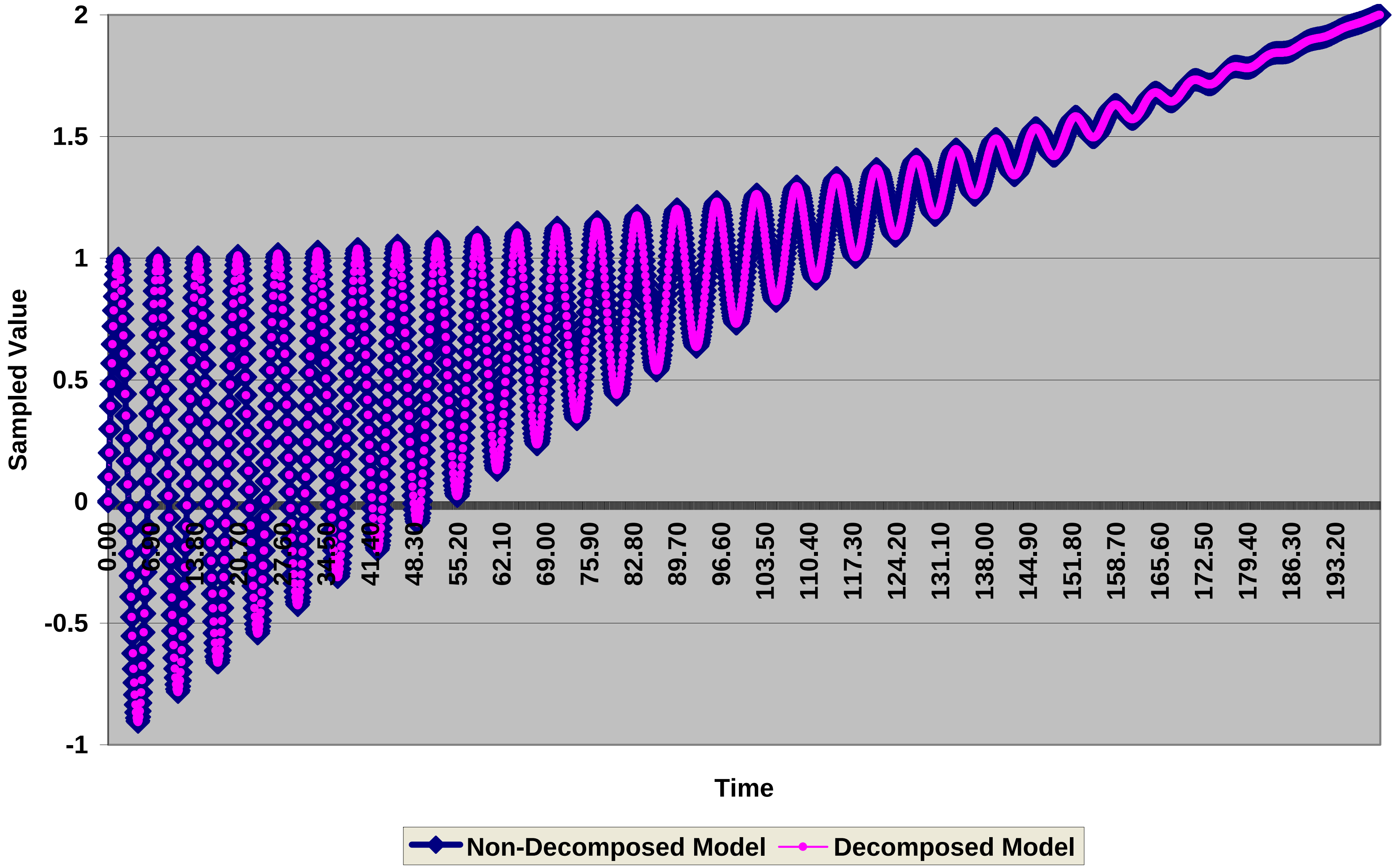
- Employ Signal



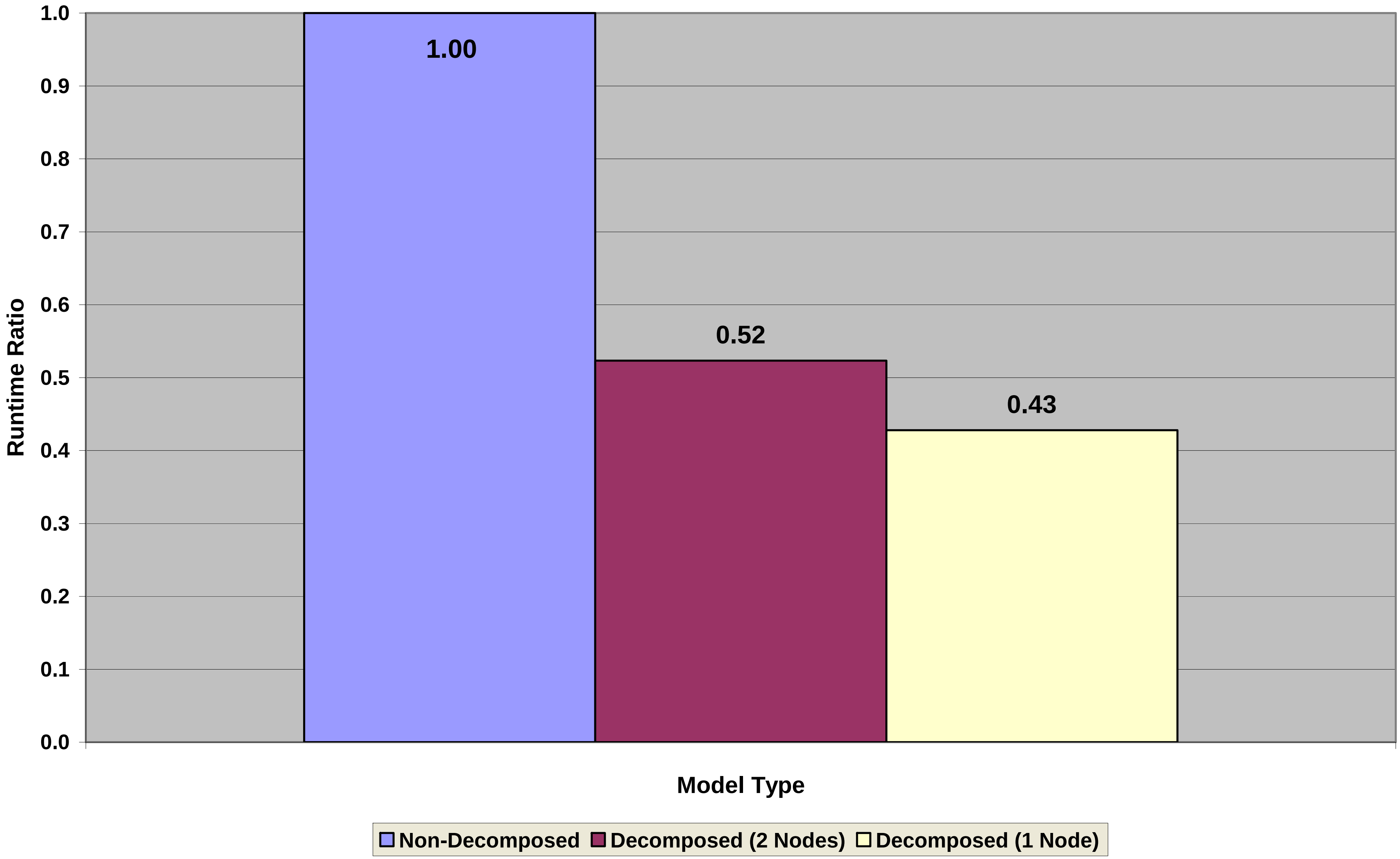
- Generate Signal



Comparing Results (non-Decomposed vs. decomposed)

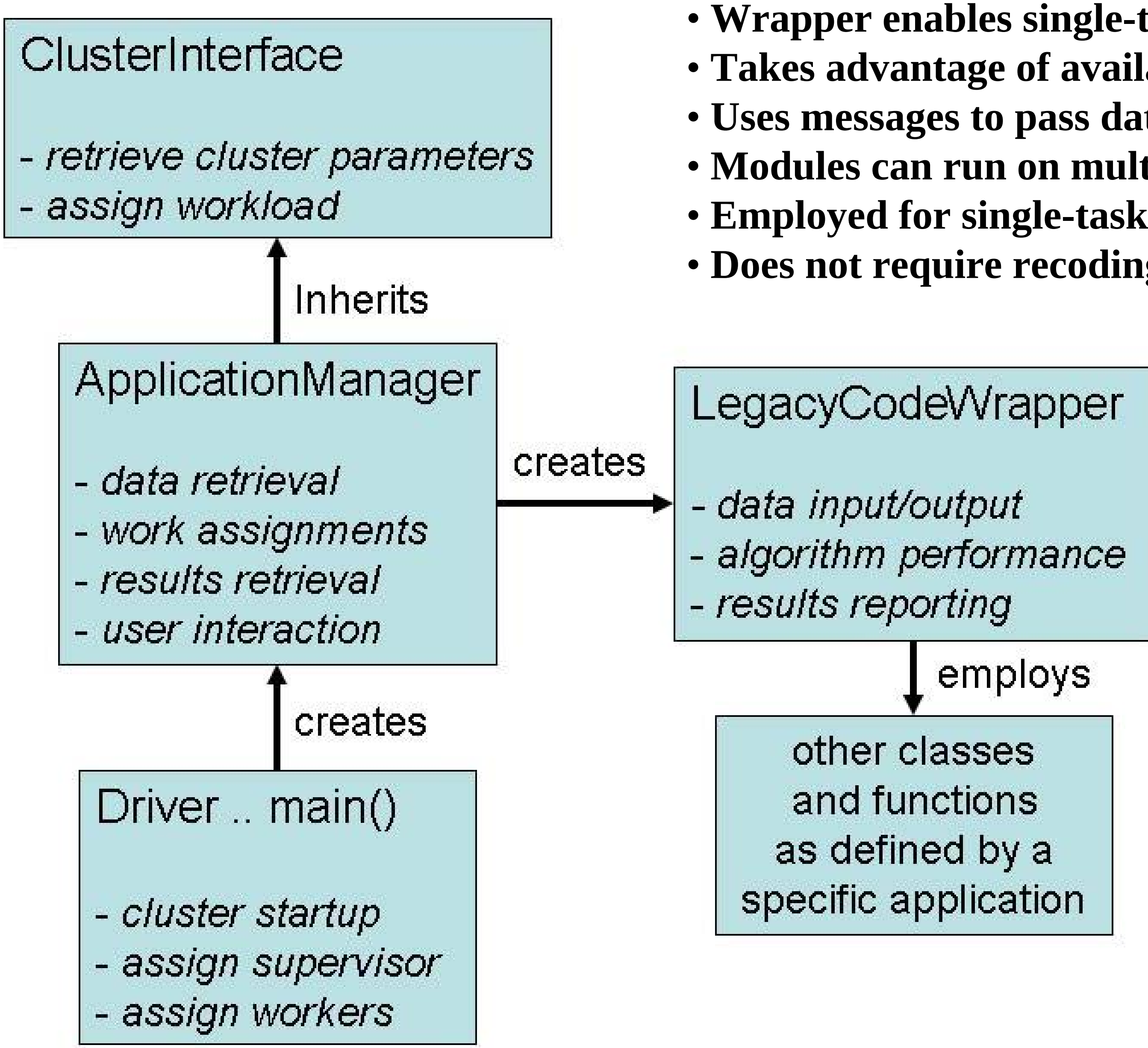


Ratio of Wall-Clock Times

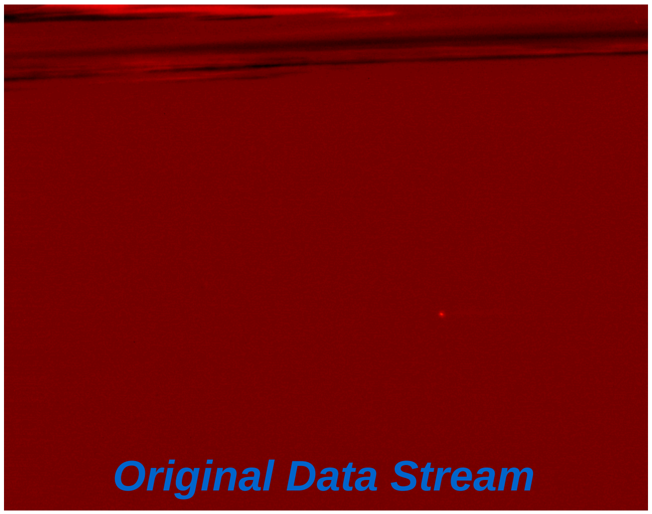


Class Wrappers for Existing Code

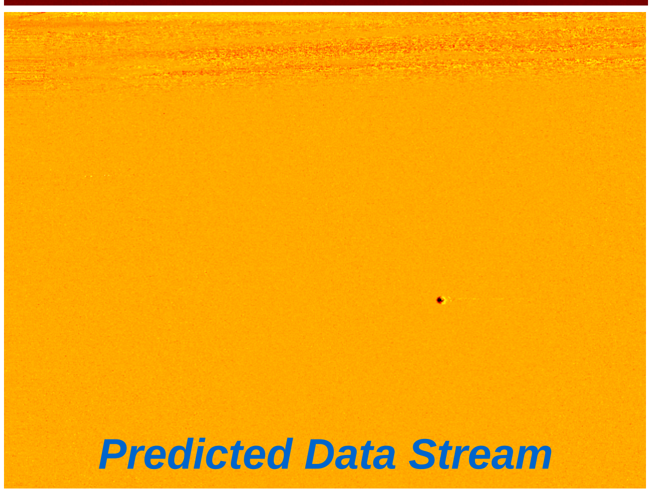
- **Wrapper enables single-task / multiple-resources**
- **Takes advantage of available cores and nodes**
- **Uses messages to pass data between modules**
- **Modules can run on multiple cores on same node**
- **Employed for single-task throughput improvement**
- **Does not require recoding to accommodate threads**



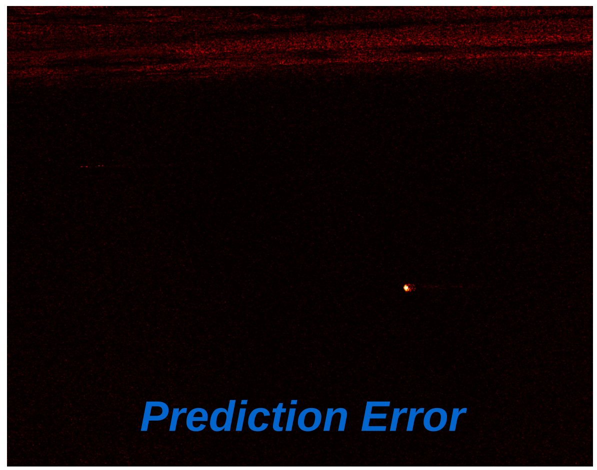
Example Application
Predictive Anomaly Detection (PAD)



Original Data Stream



Predicted Data Stream



Prediction Error

Target Nomination

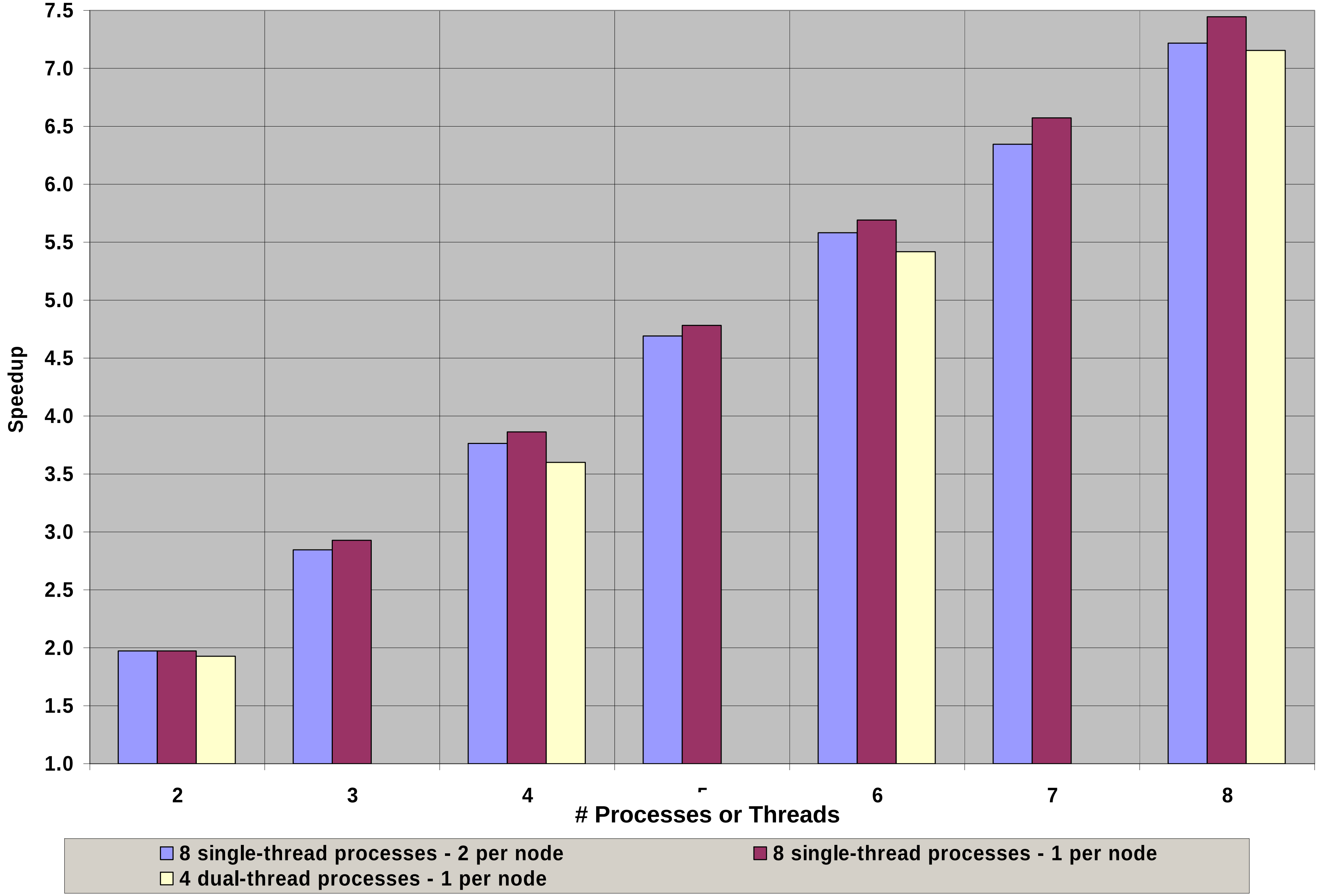
- **Target nomination in streaming data**
- **Highly generic**
- **No dependence on sensor type**
- **No physics model required**
- **Not deterministic**
- **Depends only on data temporal relationship**

Sample PAD Run

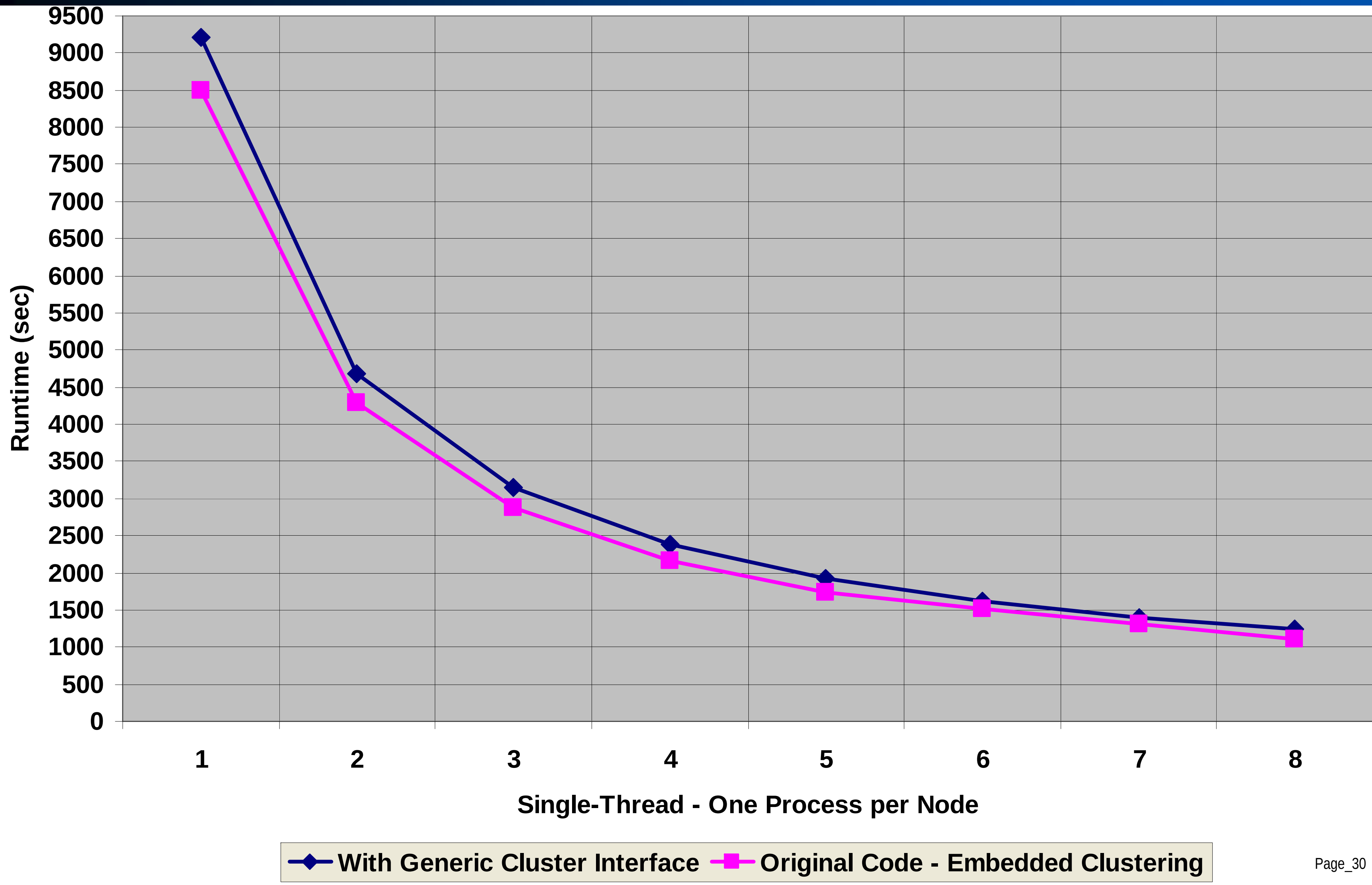
- 2200 frames (NightConqueror IR sensor, widely used in DOD)
- 512 x 640 pixels
- 60 Gaussian basis functions per pixel (19,660,800 terms total)
  - evaluate and update every frame

- Supervisor reads frame from disk
- Sends appropriate frame components to each process
- Workers process frame components
- Workers send results to supervisor
- Supervisor organizes, reports, stores results

Speedup Results
(single and multiple threads)

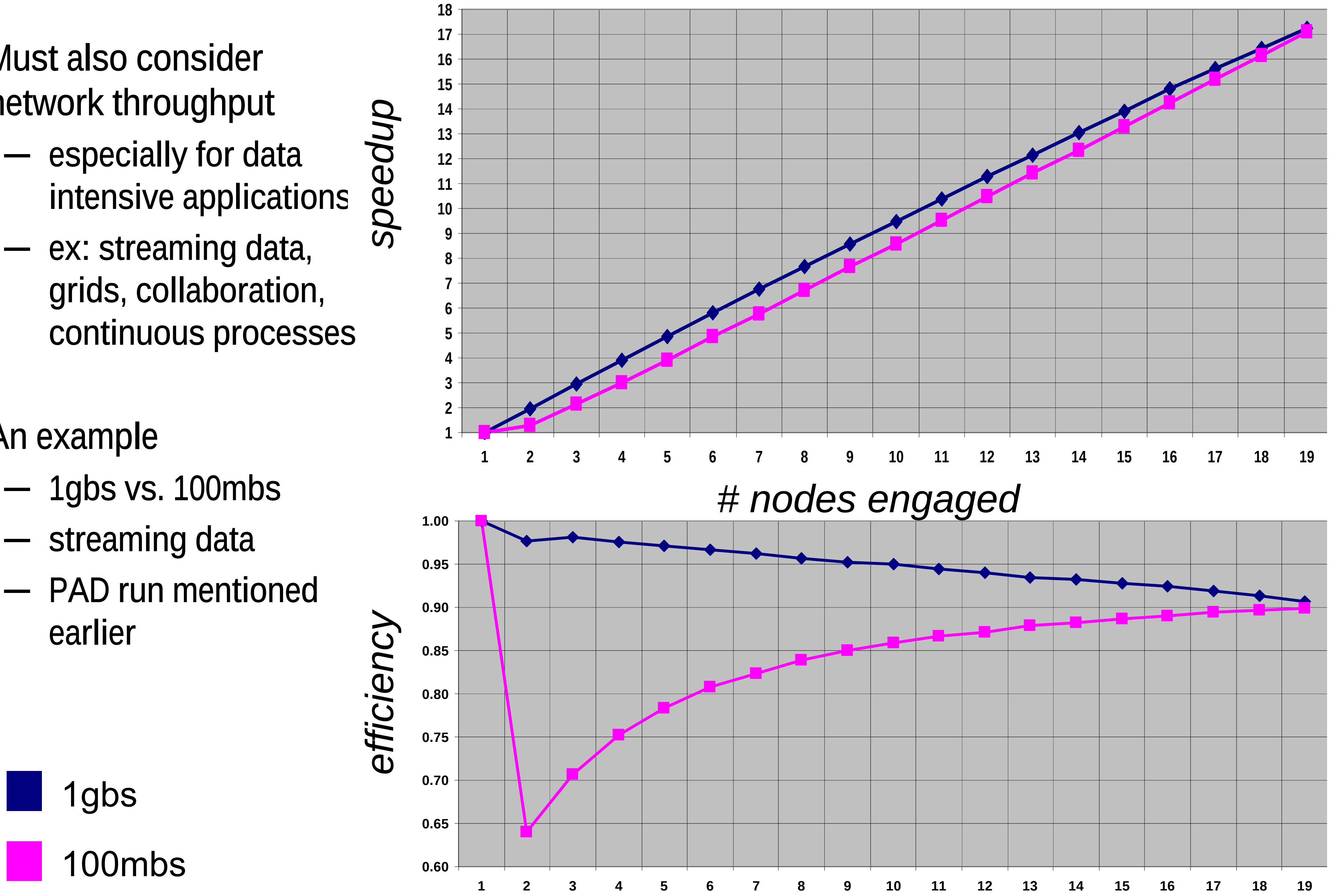


Performance Hit Using Class Wrapper



Not Just Computational Throughput

- Must also consider network throughput
  - especially for data intensive applications
  - ex: streaming data, grids, collaboration, continuous processes
- An example
  - 1gbs vs. 100mbs
  - streaming data
  - PAD run mentioned earlier



Source of Declining Throughput Gains

- Overhead
  - operating system
  - network
  - memory
  - bus speeds
  - CPU speed
  - required serial performance
  - granularity (ratio of processing vs. data passing)

Overhead can be estimated

```
f  = overhead estimation
N  = number of nodes engaged
Sn = speedup for that number of nodes
```

$$f = \frac{\frac{N}{S_N} - 1}{N - 1}$$

Amdahl (1967, 1988)
Zomaya (1996)
Raeth (2003)

Summary

- Developed philosophy for transitioning existing code to networks
- Resulted in generic approach
- Reusable - not specific to language, OS, platform, algorithm
- Dealing with application throughput, not just computational issues
- Benefits of the work
  - transition existing software to network environment without re-engineering
  - risk reduction during transition process
  - combines of network architectures with object-oriented concepts
    - but, does not require OO languages
  - employs independent, yet cooperative, processes
  - results in portable network processes
  - does not require new capital investment to yield significant benefits

Biography

Amdahl, G.M. (1967). “Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities”. Proc AFIPS, v 30, 483-485.

Amdahl, G.M. (1988). “Limits of Expectation”. Journal of Supercomputer Applications, 2(1), 88-97.

Bailey, D.H. (1991, Jun 11). “Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers”. Technical Report # RNR-91-020, NASA Ames Research Center, Moffett Field, CA.

Loger, J. (2005, Sep 7). “InnoVision Focus Areas and Challenges”. Former Director of InnoVision within the National Geospatial Intelligence Agency, briefing to the DARPA-NGA Partnership Industry Workshop, http://dtsn.darpa.mil/ixo/DARPA_NGA/images/Loger-InnoVision%20Focus%20Areas%20&%20Challenges.pdf.

Morse, S.H. (1994). Practical Parallel Computing. New York, NY: Academic Press.

Raeth, P.G. (2003). Finding Unexpected Events in Staring Continuous-Dwell Sensor Data Streams via Adaptive Prediction. Dissertation presented to the faculty of Nova Southeastern University.

Raeth, P.G. (2007, Apr). “Improving Throughput for Temporal Target Nomination Using Existing Infrastructure”. Proceedings: Intelligent Computing, Theory and Applications V; SPIE International Defense and Security Symposium.

Zomaya, A.Y.H. (1996). Parallel and Distributed Computing Handbook. New York, NY: McGraw-Hill.