

Evaluating Parallel Processing Using SMP & Workstation Clusters for ECS Science Algorithms

Narayan Prasad

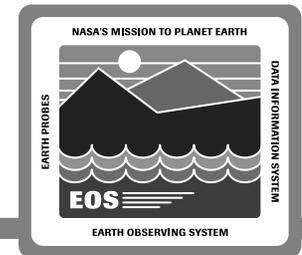
Scott Bramhall

Marek Chmielowski

Linwood Moses

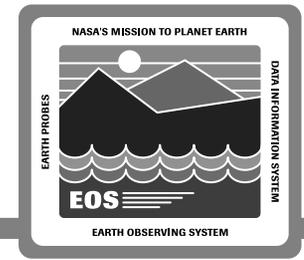
December 6, 1994

Goals

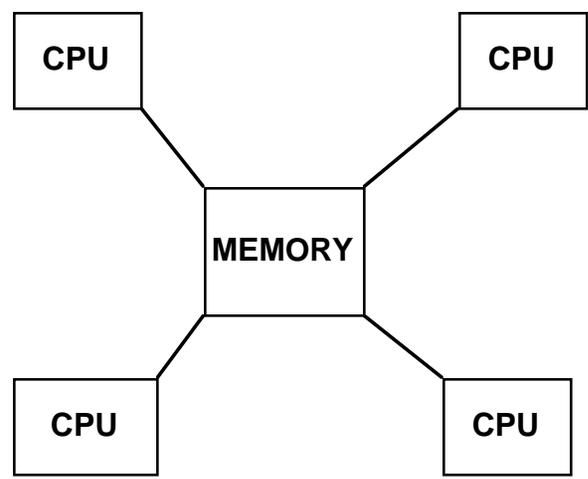


- **To determine if Symmetric Multiprocessing (SMP) & Distributed Memory parallel Processing (DMP) using workstation cluster technologies provide viable alternatives for ECS science processing in terms of price/performance, ease of creating new applications, ease of converting existing serial applications, etc.**
- **How should code be structured to extract maximum performance with minimal effort?**
- **What types of code constructs to avoid in (data) parallel programming?**
- **Evaluate automatic parallelization compilers that can be a good starting point to improve performance of a program**
- **Provide inputs to science algorithm developers who would like to use parallel programming**
- **Alert project on important issues relating to these technologies**

MIMD Parallel Memory Models



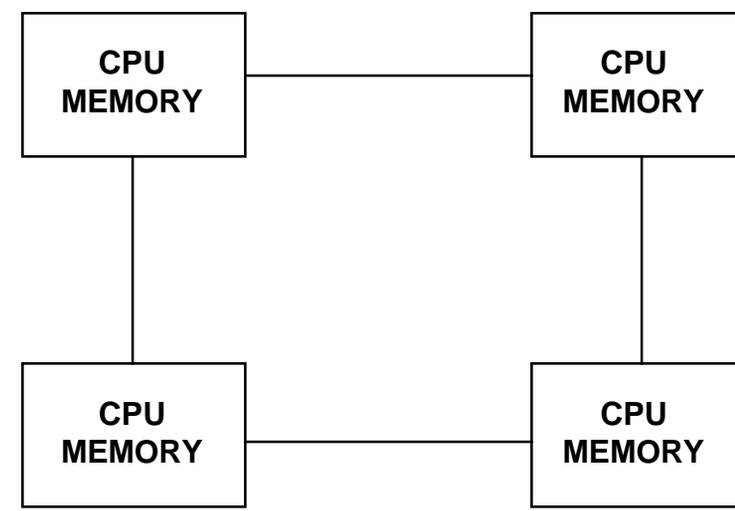
SHARED MEMORY



Each CPU can decode, issue instructions independent of other processors. All processors access the same memory.

e.g. SGI Challenge, SGI Power Challenge, Convex Exemplar

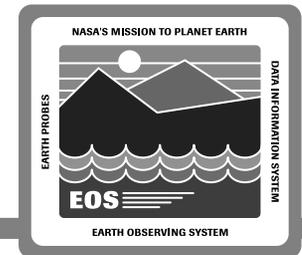
DISTRIBUTED MEMORY



Each CPU can decode, issue instructions independent of other processors. Each process can only access its own memory. Must transfer data to other CPUs if needed.

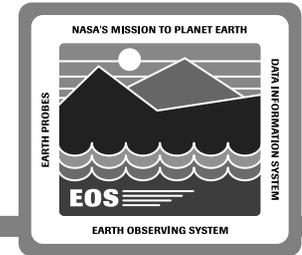
e.g. Workstation cluster, IBM SP-2, Cray T3D (can be used in shared memory paradigm also)

SMP Hardware Platform



- **SGI Challenge XL (8 processors) used for evaluation**
 - 150 MHz MIPS RISC R4400 64-bit processors
- **Power Fortran Accelerator (PFA)**
 - native source-to-source parallelizing preprocessor that enables existing Fortran 77, programs to run efficiently on SGI multiprocessor systems
 - identifies loops for data dependencies, automatically inserts compiler directives in a modified copy of the original source code
 - listing file optionally generated by PFA can be used to identify potential data dependencies that prevented PFA from running a loop in parallel
 - can freely combine serial code with parallel code for execution
- **Power C Analyzer (PCA)**
 - native C code parallelizing preprocessor similar to PFA
- **Major difference between PFA and PCA are that there are additional tools available with PFA that is not available with PCA to make parallelization easier (C is a harder language to parallelize than Fortran)**

Purdue benchmark set* to evaluate parallel processors

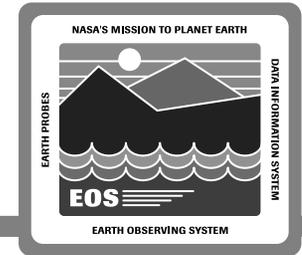


Purpose

- Evaluate constructs for (data) parallel programming
- For training purposes
- To understand code behavior in SMP and DMP environments
- Good starting point to understand parallel programming “hands-on”
- To provide insights and lessons to build new applications for parallel environments (e.g. without dependencies and other parallelization inhibitors)

*Appendix A-1 and A-2 contain more detailed information

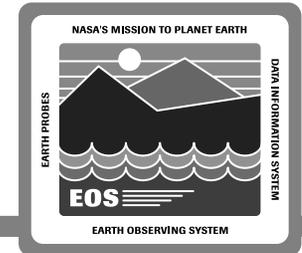
Pathfinder SSM/I precipitation rate algorithm



Background

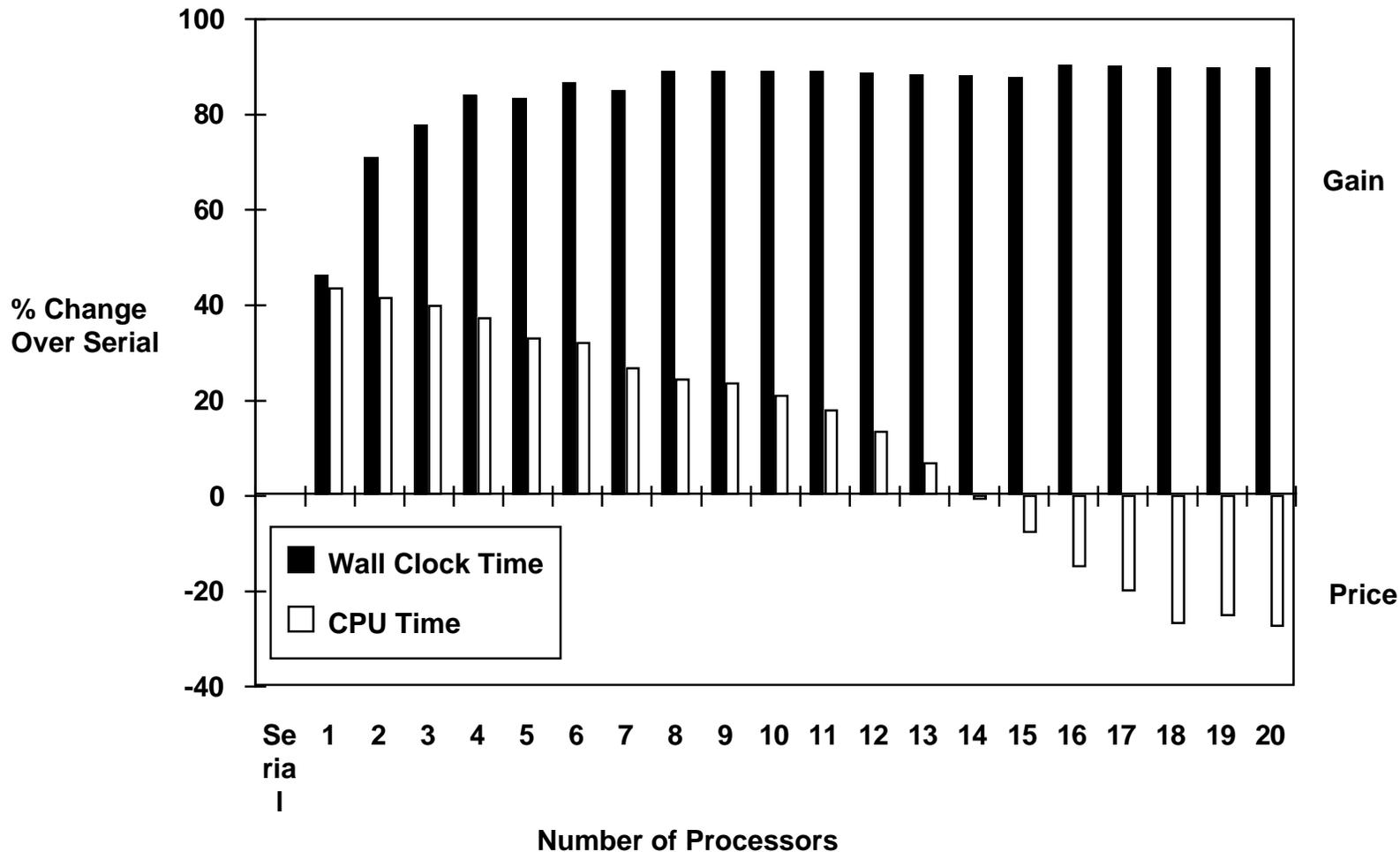
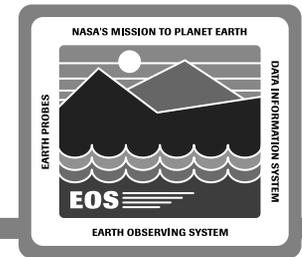
- **SSM/I instrument onboard Defense Meteorological Satellite Program's (DMSP) satellite**
 - source code from NASA/MSFC
- **Input data sets (granularized, geolocated with corrections and QC) in HDF format containing antenna temperatures, spacecraft parameters, auxiliary data sets, etc.**
- **Output data set is HDF containing precipitation rates (16 orbits/day)**
- **Algorithm**
 - processing routines to compute precipitation rates
 - routines to add geolocation to HDF file (mostly I/O)
 - routines for browse image generation
- **processing is done sequentially for each orbit for multiple frequencies**

Steps to a parallel Pathfinder SSM/I program

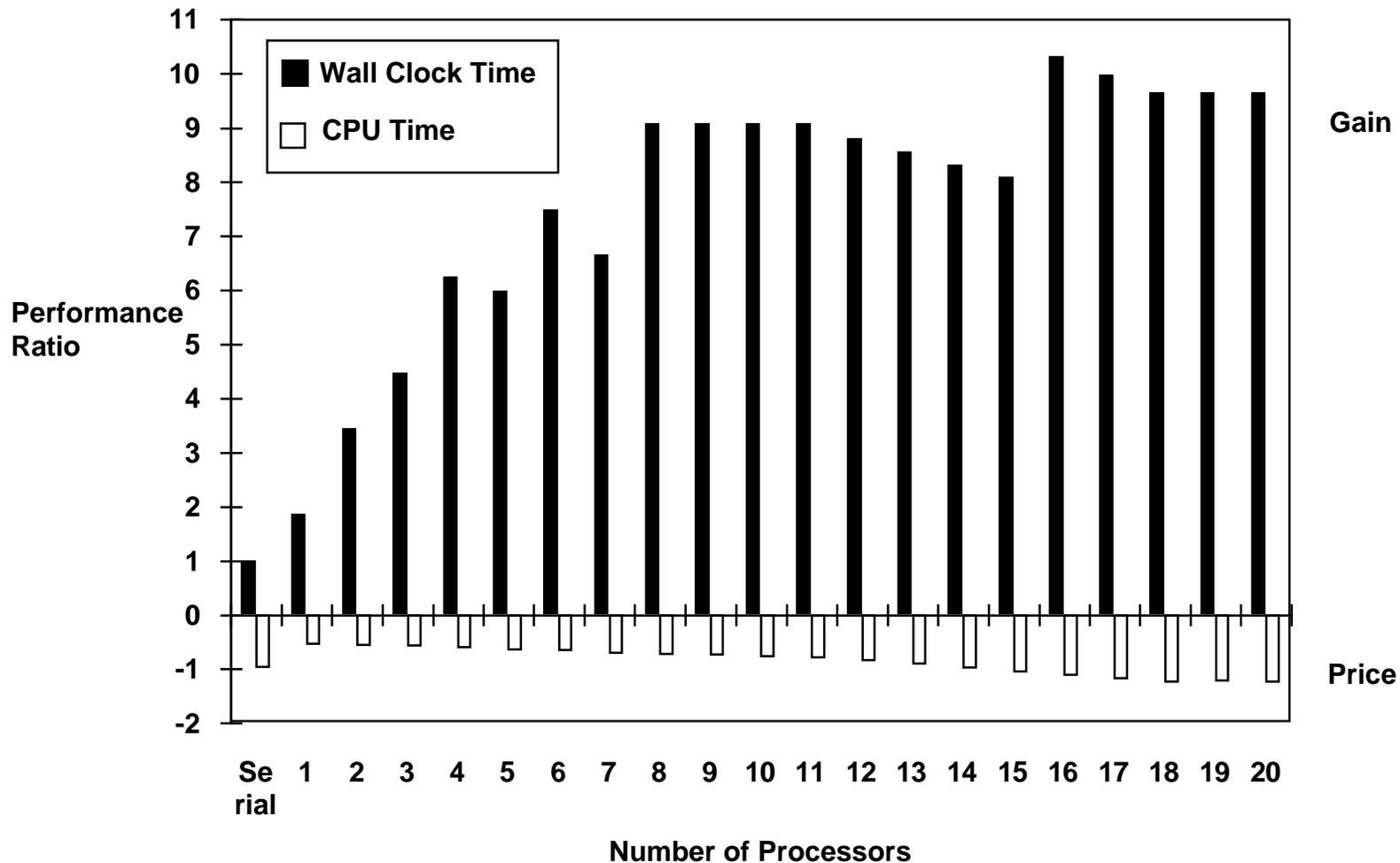
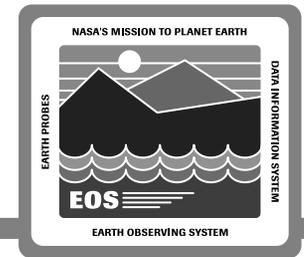


- Performance analysis of the serial program indicated that the processing part was the most time consuming
- Automatic parallelization and performance analysis indicated only marginal improvement in performance (speedup of 2 on 8 processors)
 - parallelization was never in mind during original development
- Analyzed PFA (Forge 90 from Applied Parallel Research, Inc. is also available for SMP) listing to improve performance
- Developed parallelization strategy
 - identified medium-/coarse-grained parallelism
 - found 16 orbit processing loop to be parallelizable
- Removed parallelization inhibitors
 - COMMON blocks, I/O, data dependencies, system call

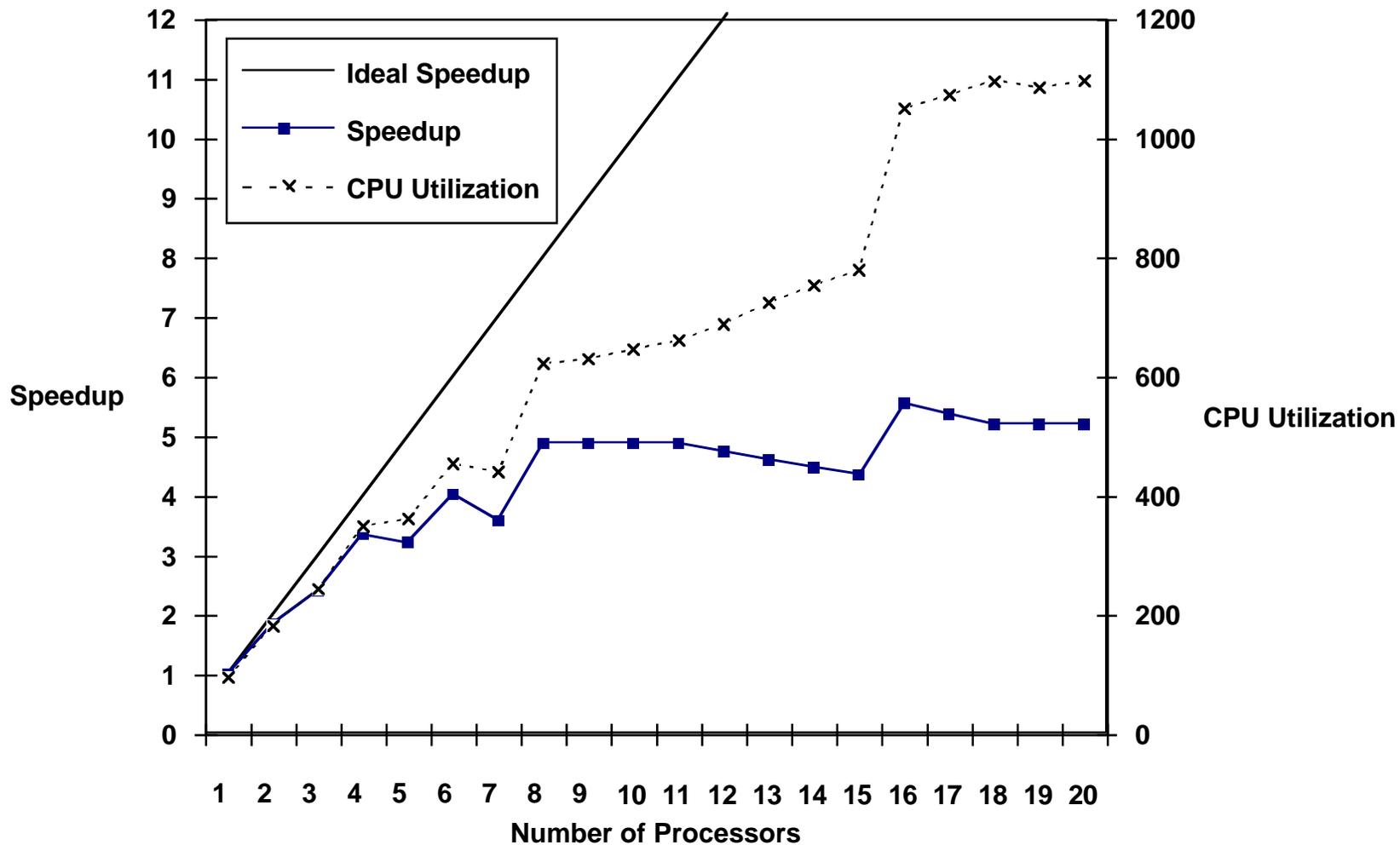
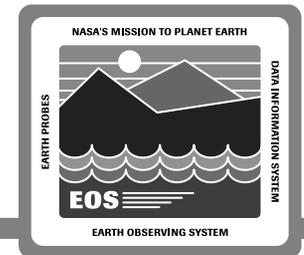
Performance expressed as % over serial (SMP)



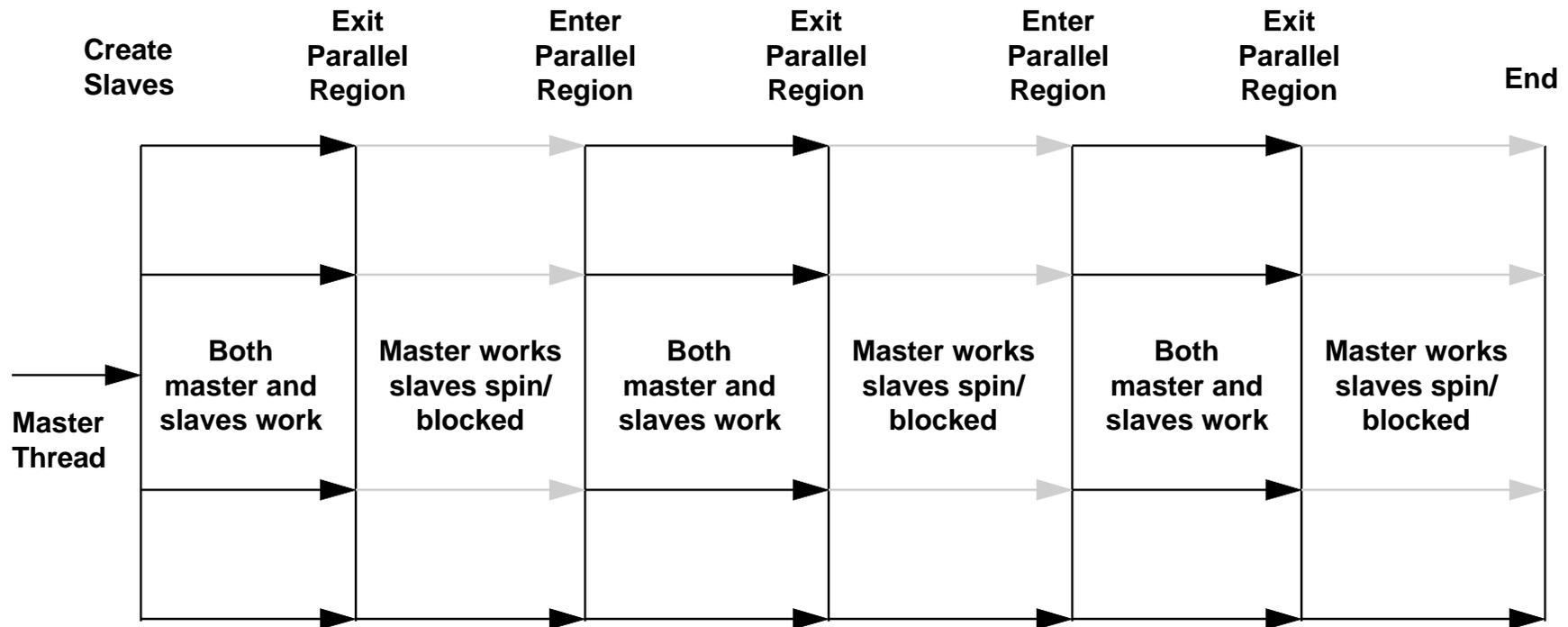
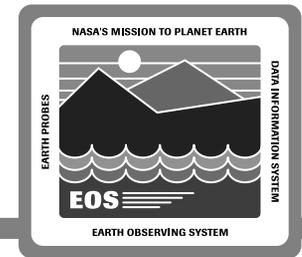
Performance expressed as ratio over serial (SMP)



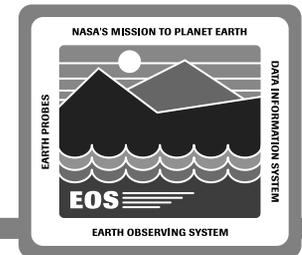
Performance as speedup (SMP)



SMP thread creation and operation

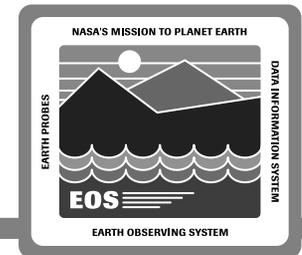


SMP Issues



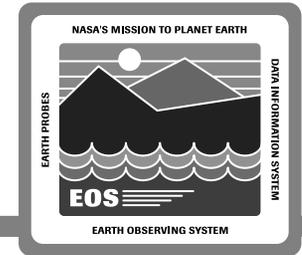
- The shared memory model allows non-threadsafe libraries to be used in serial (HDF, SDP Toolkit, etc.) by toggling between serial and parallel modes - Refer to Technical Paper # 194-430-TPW-001 in the EDHS SDPS bulletin board (see Appendix A-5)
- This can affect overall performance of a parallel algorithm, and may not work for all algorithms. Some algorithms may require use of toolkit functions within a parallel region (e.g. geolocating pixels/cells when processing pixels/cells in parallel)
- Potential impact of SDP Toolkit on parallel processing
 - Shared memory model requires special handling of global address space
 - All toolkit functions have I/O in their calling chain (for status messaging, etc.). This inhibits parallelism (unless designed for parallel I/O when standardized). For now, cannot use any toolkit function inside a parallel region because of embedded I/O.
- Potential impact of HDF libraries on parallel processing
 - provide some parallel I/O support on select machines like CM-5
 - Shared memory model requires redesign of global address space. For now HDF can only be used serially on most machines.

SMP advantages/disadvantages



- **Advantages**
 - **SMP is easier to program (1.5 MM for SSM/I < 2000 LOC) - to be interpreted with CAUTION!!!**
 - **Good for programmers who like reasonable performance with minimal effort**
 - **No load balancing problem - performed automatically by the operating system**
 - **Source code is portable (no machine specific instructions)**
 - **Achieving high performance by parallelization is relatively painless**
 - **Tool intelligent enough, shifting a fair amount of responsibility away from the programmer**
- **Disadvantages**
 - **SMP is not very scalable (DMP is scalable)**
 - **I/O is serial**
 - **Key to successful parallelization still lies with how well the analyst understands the algorithm, and his ability to abstractly identify parallel and independent portions within the algorithm, and the skill of his programmer to code without introducing parallelization inhibitors**

Parallel computing on workstation cluster

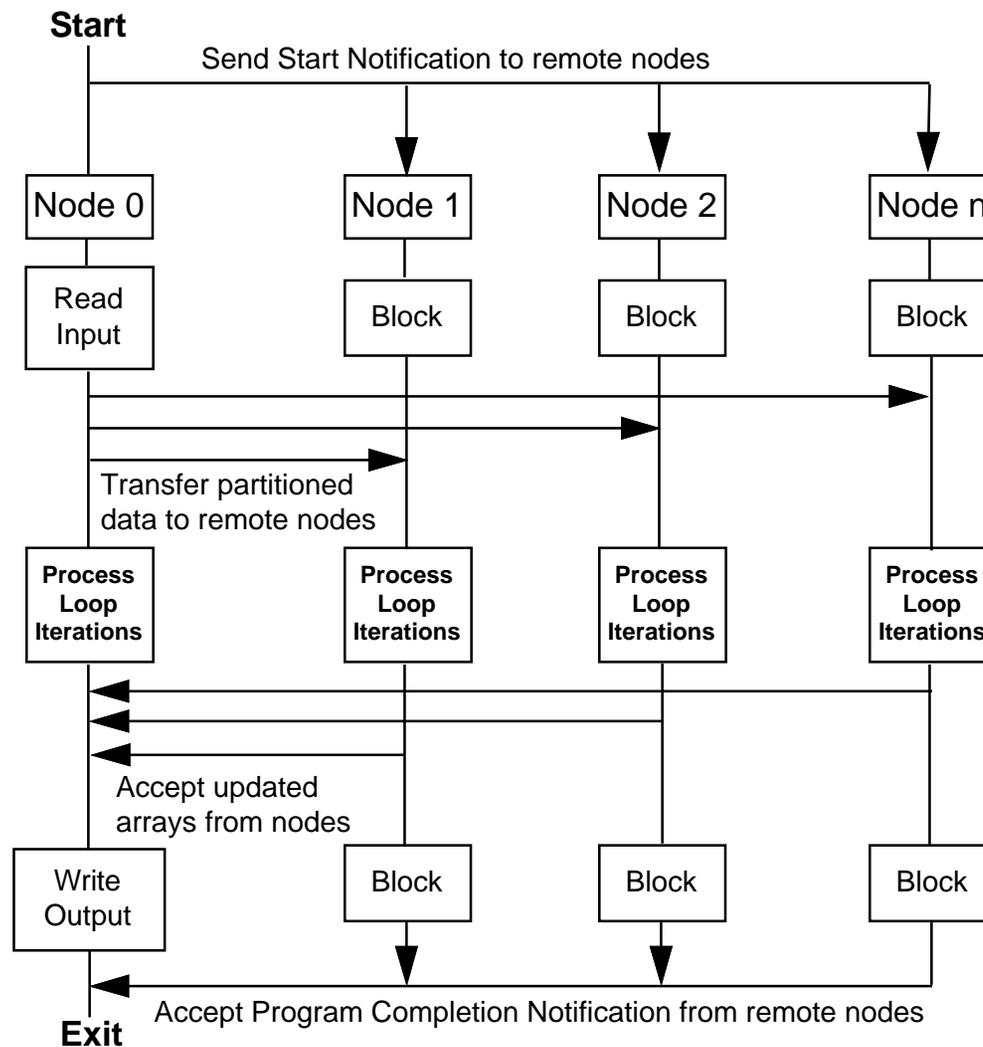
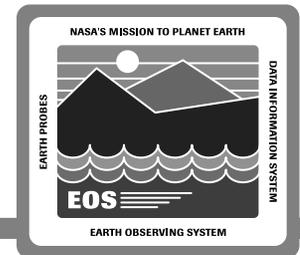


Distributed Memory Parallelizer* (Forge 90 and xHPF from Applied Parallel Research, Inc. [APR])

- **SPMD (Single Program Multiple Data) model (a natural extension of sequential processing)**
 - all processors execute the same program
 - iterations of parallelized array operations and DO loops are distributed across processors on a network
 - they work on different data depending upon allocation and distribution of arrays referenced within the enclosed code
 - serial code and undistributed loops are replicated on all processors
- uses APR and subset HPF directives
- uses a message passing library for communication (PVM, Linda, etc.)
- I/O is performed on one processor designated as processor zero

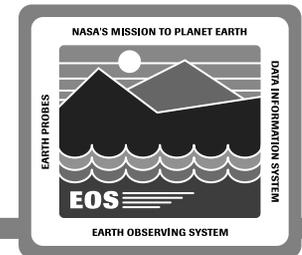
* See Appendix A-3 for more information

SPMD Model



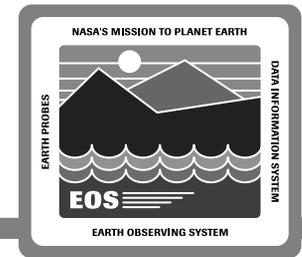
All Nodes
execute the
same code

APR Tools vs. I/O Libraries



- When I/O statements are specified as Fortran “Read/Write” and compiled using auto parallelization compiler, then I/O is automatically understood and performed on node 0 (serially because I/O is not currently parallelizable)
- When called functions are in another language like C (e.g. SDP Toolkit, HDF), the code is replicated on all processors and the presence of I/O statements are not understood by the parallel compiler. The replicated I/O function is then executed on all the nodes which can cause failure (when writing output).
- How do we handle I/O with HDF/SDP Toolkit?
 - Solutions/workarounds to handle I/O for HDF
 - Write jacket routines to ensure I/O is performed only on node 0 (originally proposed by APR - not the recommended solution because it is too complex). However, it will work for any kind of data arrangement within a file
 - Processes running on multiple processors can read appropriate portions from the same file simultaneously (because they have unique process id and have separate address space). For output write in serial mode (easiest to implement but can only be used for input that has already been preprocessed with data arranged as orbits, swaths, cells, etc.)
 - Use parallel I/O when available in standardized form for both input and output (APR tool appears to be poised for parallel I/O) - most elegant solution
- No solutions for SDP Toolkit at this time

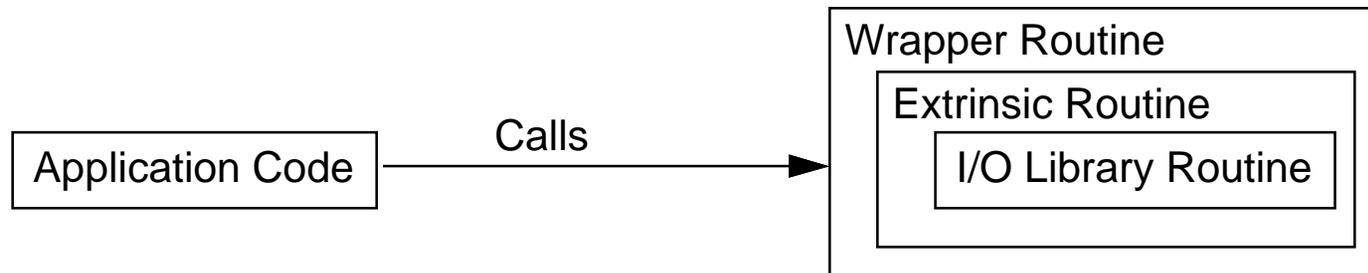
Implementing calls to I/O libraries



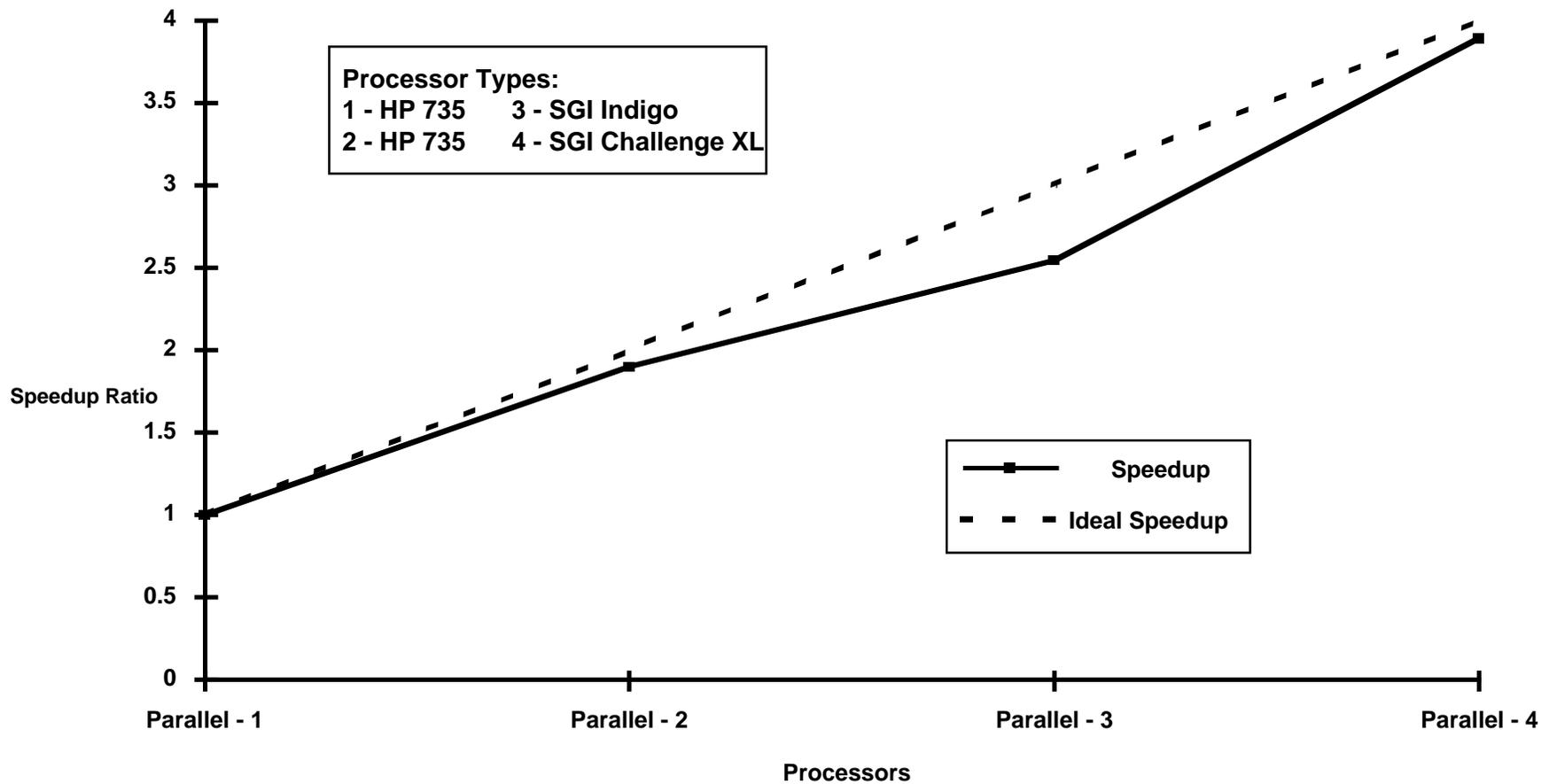
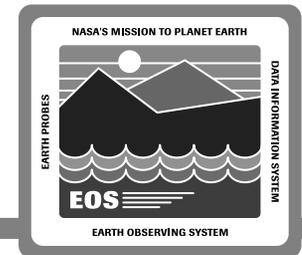
- **Serial implementation**



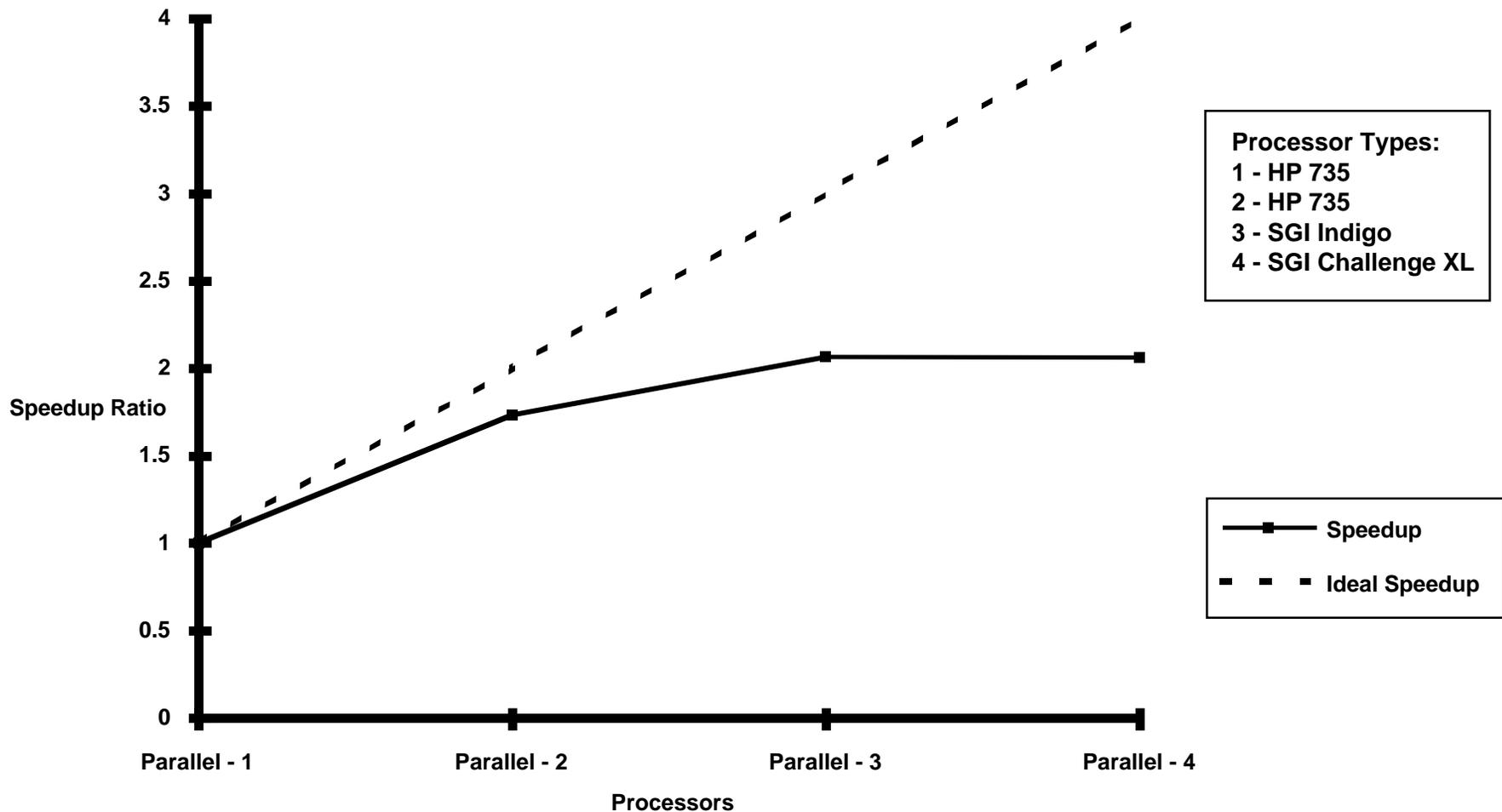
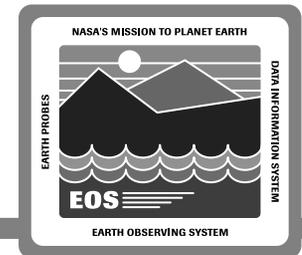
- **“Wrapped” Parallel Implementation (see Appendix A-6)**



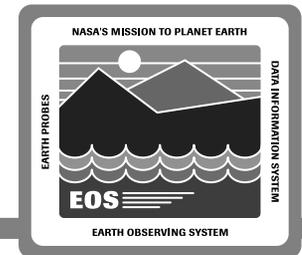
Speedup (processing) on workstation cluster



Speedup (processing + I/O) on workstation cluster



DMP Workstation cluster



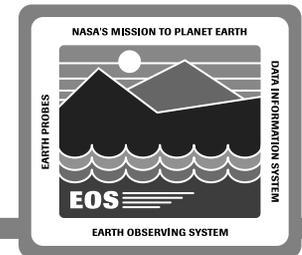
Advantages

- Economical
- Source code is portable
- Dynamically scalable (can work even on a single workstation)
- Can be used as a testbed for parallel program development for MPPs
- No address space contention because of separate memory (one less worry for HDF, SDP Toolkit)

Disadvantages

- Handling I/O
 - parallel I/O from hardware perspective (limited support on few m/cs like CM-5, etc.)
 - parallel I/O from software perspective (PASSION from Syracuse University, etc.) - some implementations but no standards available
 - SDP Toolkit will need special handling of I/O, process control file, global buffer management, etc. in a DMP environment
 - parallel interface not available for HDF operating in workstation cluster environment
- Handling failure and recovery is more complicated than DMP in one cabinet (IBM SP-2, etc.)
- Overall performance is dependent on interconnect performance (very high performance networks can make it seamless)
- Must use other means for dynamic load balancing

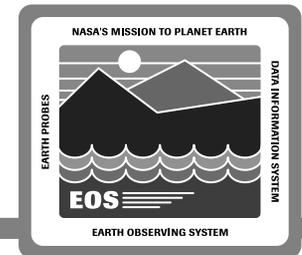
DMP Workstation cluster (cont.)



Other issues

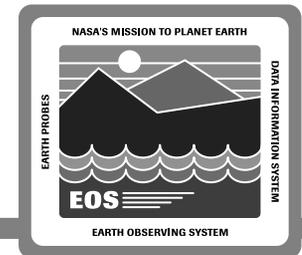
- DMP tools are advanced for Fortran but not so for C family
- Heterogeneity (must be of same architecture family when using tools)
- Parallelization tools for SMP clusters are also available combining shared and distributed memory paradigms to address multiple levels of parallelism in a single portable program

SMP and DMP summary



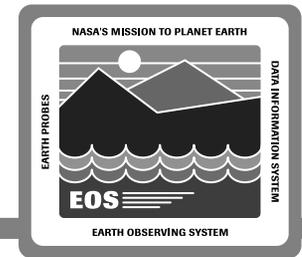
- **A parallelization strategy should be done at design and implemented when coding. A parallel program will run on a single processor also.**
- **Both SMP and workstation cluster technologies are viable alternatives to serial computing**
- **Parallelization tools are very sophisticated. Can guide even novice parallel programmers to set up applications quickly.**
- **Parallelization tools to cluster SMPs together as SMP/DMP should further make cluster computing attractive and easier on the programmer**

Lab Demo



- **Pathfinder SSM/I on SGI Challenge (SMP)**
 - **serial and parallel (with and without SDP Toolkit) versions will be compared**
- **Purdue benchmark set on workstation cluster demonstrating distributed heterogeneous cooperative computing**
 - **7 testbed workstations of different makes will be used**
- **Pathfinder SSM/I on workstation cluster**
 - **Resource restriction permits distributed computing on only 4 platforms**

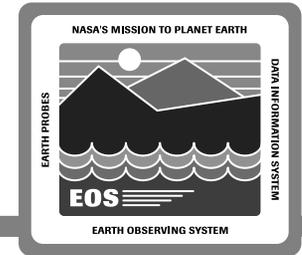
APPENDIX



The following slides are for additional information and will not be covered in this presentation

A-1

Purdue benchmark set to evaluate parallel processors

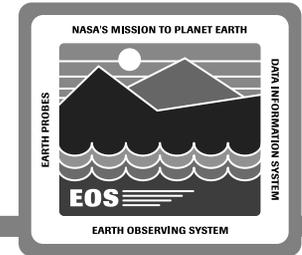


Background

- **Purdue benchmark set is a collection of computational problems, that are simple, yet diverse, and selectively address different aspects of parallel computing**
- **Suite of 15 problems extracted from larger computations representing a sample of practical computations**
- **The problems represent various schemes of data dependencies, from very simple application with almost no interprocessor communication, to more complex with large communication overhead, to irregular problems difficult to parallelize**
- **Benchmarks are available in Fortran 77, Fortran 90, Fortran 90D and Fortran for Massively Parallel Processors using message passing**
- **Used Fortran 77 benchmarks for this evaluation**

A-2

Purdue benchmark set to evaluate parallel processors

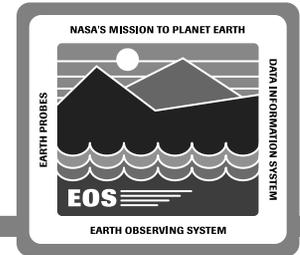


Summary

- How the different problems parallelize to the PFA are analyzed to gain understanding of the parallelization preprocessor
- PFA (or PCA) appears to be a good starting point for parallelization of sequential algorithms
- Good performance with minimal effort depends upon how well the code has been thought about originally
- Parallelization strategies provided by the PFA are useful to further “hand code” automatically parallelized programs to improve performance
- Analyzing Purdue set behavior provided insights into parallelizing a real world application - Pathfinder SSM/I precipitation rate algorithm.

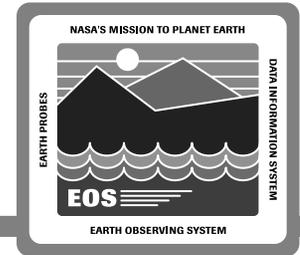
A-3

APR Toolset

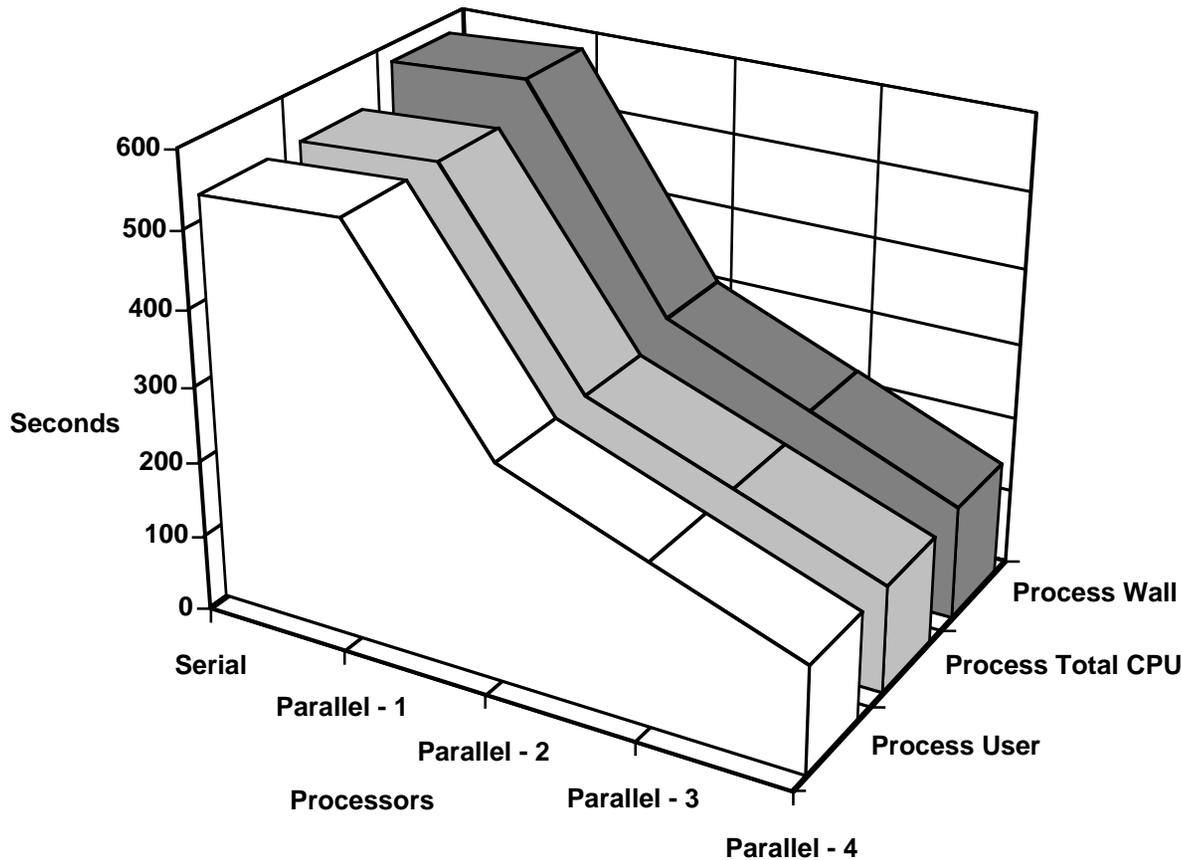


- **xHPF**
 - **Fortran 77, Fortran 90 batch parallelizer**
 - **can use directives for user-customized parallelization**
 - **can use serial timing statistics to better tailor automatic parallelization**
- **Forge 90**
 - **Same as XHPF except that it is a GUI-based interactive Fortran analyzer/parallelizer**

Performance of SSM/I on workstation cluster



SSM/I Main Process



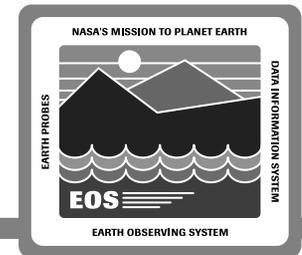
Processing part was repeated 10 times for 16 orbits of data

- Process User
- ▒ Process Total CPU
- Process Wall

- Processor Types:
- 1 - HP 735
 - 2 - HP 735
 - 3 - SGI Indigo
 - 4 - SGI Challenge XL

A-5

Documents on prototyping

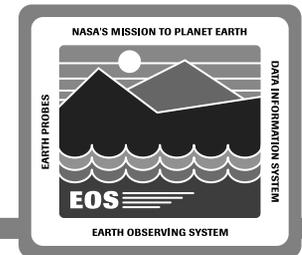


The ECS Data Handling System (EDHS) provides information about all the prototypes. The URL is <http://edhs1.gsfc.nasa.gov/>

- Click SDPS under Segment Office Home Pages
- Click SDPS Prototypes
- This prototype is listed under Science Software Execution Prototype

A-6

Jacket routines implementation



- The wrapper declares new versions of arrays to hold input variables and passes them to extrinsic. The wrapper also distributes data returned from extrinsic
- Extrinsic determines if the node running is node 0, and if so performs I/O. Returns input data or status to wrapper