Support further progress in the fields;
advance, replace, and even allow experimentation to
scientific and engineering needs for simulation to
more complicated functions;
increasing economic markets for consumer devices with
computational devices at many economic levels;
an increase in the performance of computers and other
in recent years due to:
The use of computational devices has increased tremendously.

What is Computational Science and Engineering?
devices.

and placed more emphasis on simulation and intelligent

political concerns that have precluded certain activities.

ambitions have exceeded computational capabilities,

engineering applications where application/algorithms

•
science and engineering applications.

The development of these strategies applicable across multiple forms of technical constraint. The approaches taken stress that are currently not solved due to some engineering devices to solve problems in science and computational devices to solve problems in science and engineering.

**Definition:** Computational Science and Engineering
of computer scientists.

to do so. (The latter bit is for the most part the concern
inventing and implementing the infrastructure necessary
use computers, e.g., new algorithms and models, but also
includes not only inventing and implementing ways to

algorithms, discipline, e.g., computer architecture and numerical
discipline, e.g., computer science, and multiple specialties within a
multiple high level disciplines, e.g., meteorology and
CSE is multidisciplinary at many levels, i.e., it requires
communications
  • signal processing
  • control of systems
  • electromagnetic simulation

distilling and exploiting work in (1).

constrainted practical problems. (This often involves

II. Engineering: Develop algorithms and systems to solve

universe and their interaction for basic science.

modeling the various parts (real or imagined) of the

and to analyze.

I. Science and Applied Mathematics: Develop and interpret

Who does what?
architecture and construction

manufacturing and robotics

aerospace design

sensing and imaging

design of circuits and devices
III. Computer Science and Engineering

- Hybrid Architecture: General purpose, special purpose,

- Algorithms: numerical and nonnumerical

- Software:

- System: runtime libraries, OS, languages,

- Code generators/compilers

Design tools: prototyping environments and utility codes

- Performance evaluation, restructuring compilers, performance evaluation

Use tools: visualization, databases, problem solving

environments

- Systems
- computer and circuit testing
- catastrophe simulation
- nuclear weapon simulation
- car crash simulation
- economic or political expense replaces experimentation at least partially due to
- particle detection
- predicts results to guide the design of experiments
- extends range of parameters - high speed air flow augments experimentation -

In I and II computation
III is mostly concerned with the tools needed to do I and II

- black hole behavior
- collision of galaxies
- precious, are too dangerous!
- interaction of chemicals that do not exist; are too

allows experimentation •
orbital computations

- celestial calculations (evolution of galaxies or stars)

- geological modeling (continental drift)

- climate modeling (global warming or cooling)

Simulate much faster than real time

Usefulness of results:

Constraints relative to real time in simulation affect the

Constraints and Computing
action for a problem in real time
fault simulation in power grids (deterministic corrective
weather prediction
- Simulate faster than or close to real time
- "Fast" physical systems, e.g., chemical reactions
- Simulate much slower than real time
look for more efficient implementations on large machines

•
look for more powerful machines

•
look for more sophisticated models

•

Trends:

the activity

Refining the model and applying to do science is the point of

and the speed of the computational platform available.

These are typically constrained by the accuracy of the model.
Gene mapping

controller design

3. Parameter evaluation or pattern detection

hours to simulate

a few nanoseconds of a circuits operation can take

2. Time response of a system

heat profile of material

or antenna pattern

Electromagnetics computation of radar cross section

I. steady state computations for off-line design

scale directly

usefulness of results, i.e., not relative to application real time

Time constraints that affect efficiency of design process not
cost of evaluation

develop a hierarchy of models where accuracy varies with

look for more powerful machines

look for more efficient implementations

look for more sophisticated algorithms

Trends:

the point of the activity

reasonably well understood and its application not tuning is

An operational platform available. Model is usually

model and algorithms to get a desired accuracy and the speed

These are typically constrained by the complexity of the
missile tracking and avoidance
remove atmospheric turbulence
mirror control on optical observation systems to
robotics, 
VLST manufacturing
takeoff, flight, landing of aircraft

2. adaptive control
jammer suppression

echo cancellation
noise suppression

1. adaptive filtering algorithms

implies severe computational limitations.
Time affects the usefulness of the results but the application
Knowledge that may or may not be undergoing updates.

4. Many of these algorithms also involve stochastic optimization, i.e., working with incomplete environment.

3. Intelligence gathering (pattern detection, image analysis)
more efficient ways of assessing the interaction of the two.

- development of specialized purpose devices
- more efficient computational platforms (often drives)
- more sophisticated algorithms
- computational and I/O loads.

Data intensive applications are increasing the absolute terms but not relative to computational platform. However, for these types computational load may be low in absolute
(engineer, applied math)
Better tools to do all of above (computer science and
and engineering)
Better machines: big and small (computer science)
(applications)
Better algorithms: numerical and nonnumerical
Better models: continuum and particle (science and

What do we want?
scenario, it may or may not change the algorithm.

sampled image – improves fidelity to observed

Need for larger problems, e.g., more points in a

model; it may or may not change the algorithm.

fidelity to given model but does not improve the
discretization of a continuous model – improves

Need for larger problems, e.g., more points in a

systems is the main tradeoff.

Application/algorithmic ambition vs. Affordable

High performance is always a relative term.

Summary for Parallelism/High-performance

•
cost for current processing systems.

- Improves capabilities but may have too large of a

  - Improved communications/Signal processing

Need for more sophisticated algorithms, e.g.,

- Inhomogeneity, hopefully improves science.

• Increases computation, tends to introduce

- Sophisticated physics in an atmospheric model

Need for more sophisticated models, e.g., more

•
Performance Models and Metrics

Algorithms and Applications

Software

Architecture

Discussing parallel or high-performance processing
Issues from 4 different areas must be considered.
network design

memory system design

processor design

Three main components to consider:

- Identified.

The limitations of these assumptions must be identified.

- The performance improvements assuming assumptions upon which the performance of architecture must be identified alone.

Key features of architecture must be identified alone.
different architectures.

mapped to low-level SW in different ways on compromise with more generic constructs that can be

Higher-level SW support (Fortran, C, C++) tries to

available.

used to make explicit control of specialized features

Low-level SW support (assembler level) tends to be

consistent

performance must be identified (they are not always

available and those required to achieve high

levels of software / architecture interaction

Software
Typically done in the name of portability.

underlying architectural assumptions – this is programming paradigm that contradicts the details of the architecture completely and using a

This can be carried to the extreme of masking the •
Critical performance problems
Support libraries are crucial to isolate and solve

• Effective design and use of computational and

not be available to the generic user.
Constructs are available on the machine, they may

• Even though low and higher-level software

significantly affect the approach taken
associated programming paradigm choice can

• Explicit vs. Implicit parallel programming and the
to demonstrate this

we will use examples from numerical linear algebra to

(both is typically the best)

purity software manipulation point of view (using

than dealing with the tuning of the code from a

often be more effective when optimizing performance

computational primitives and their relationships can

use of the mathematical characteristics of the

●
of new systems

significant barrier to fully exploiting the capabilities

the requirement to use legacy code is often a

- parallel code (yours or legacy)

- parallel algorithm

- serial code (yours or legacy)

- serial algorithm

Examples:

techniques available and the difficulty of the process

high-performance code significantly affects the

The starting point from which you proceed to develop
things

fairly grim (especially on the commercial end of
subject of much research but the situation is still
similar tools for parallel systems have been the
•
tools are fairly well-developed.
sequential debugging and performance monitoring

the performance of any code.
Performance monitoring tools are critical to tuning

by the system software and not the user.
restoring compiled code to parallel code that is seen and compiled only
• often systems require the use of support tools such as
The ultimate measure of success is given by the algorithm/application community.

While theoretical and aesthetic beauty in parallel algorithms are satisfying – fast, accurate and robust algorithms solving real application problems are better.

Merging architecture and software considerations is the most difficult part of parallel programming.

Algorithms and Applications
to be expensive.

to actually implement both and time it. This tends
accurate model of an algorithm on an architecture is
and cost of evaluation for these model, i.e., the most
Typically a tradeoff exists between level of accuracy

- Performance models - Typically combining empirical

- and theoretical forms - accomplish this.

- architetcure/algorithm design and tuning.

- The interaction of the architectural/software

Performance Models and Metrics
defeat.

required to assess progress and declare victory or architectural and algorithmic parameters are also considered acceptable performance as a function of

Performance metrics that emphasize what is •

tractability: Some simplification is required for the sake of •
Rules of Thumb for Efficient Processing

1. Do not wait until one thing is finished to start something else.
2. Do not do more work than you must.
3. Do not let the cost of following Rule 1 or Rule 2 outweigh the benefit of following Rule 1 or Rule 2.
Some Important Considerations

- Efficiency — Efficient execution may alter stability and correctness of the algorithm. Sometimes this is not easy to test.
- Correctness — Efficient execution should not destroy the correctness of the algorithm due to altered search paths.
- Parallelism can also affect the quality of the solution of optimal algorithms that have suspect stability.

There are many computationally difficult to check. There are many computationally slower and more reliable algorithms. (This can be very roundoff properties of numerical algorithms compared to

Accuracy —
SW support of the machine.

- In architecture you are considering maps well to the HW and SW architecture (HW and SW).
  - Be careful that the technique you are considering maps well to the HW and SW architecture (HW and SW).
  - Forgetting it.

- Time it takes longer to figure out that you can save work than the time it takes to do the work you save then.
significantly different in SW or HW architecture. Performance percentage on another machine that it the mix. The resulting code will not achieve the same implementation of your code, you have to customize for precisely for a machine running the best possible portion of the peak possible for a machine or more algorithm and library design, etc. To get a substantial • Portability — The classic dilemma for high-performance is not however very useful. so to do.

• Generality — It is very easy to build a very fast program Considerations (cont.)
coarse grain – hundreds to thousands of

iteration of a loop

processor as basic work unit, e.g. a single

fine grain – tens of instructions executed by each

system

across multiple processors in a parallel processing

simultaneously

multiple machine instructions running

functional unit parallelism – very fine grain, i.e.,

multiple sub-operations running simultaneously

pipelining – extremely fine granularity, i.e.,

We will consider parallelism within a single processor •
combinations of algorithms
multiple programming paradigms, and adaptable

tends to exploit multiple levels of granularity,
produce a large system — clustered computing
to and across multiple parallel processors combined to

the user's job
subroutines that make up a significant portion of
work unit — more complicated functions or
instructions executed by each processor as basic
Characterizing Systems

- architectural paradigm
- programming paradigm

Two key axes
hybrid

- distributed memory

- shared memory

- vector/array/data parallel

- data flow

programming paradigm
control (scheduling) / synchronization support
communication / coherence support
network topology
(memory banks)
memory-based (relative position of processors and
control-based ( Flynn)
taxonomies
architectural paradigms
in practice, this is not always true.

For simplicity, it is assumed that there is only one process/thread per processor (and HEP).

and some earlier machines such as Alliant

system and some earlier machines such as Alliant

instructions (HW supported threads, e.g., Cray Tera

processes (Posix threads) to a collection of a few

threads range in complexity from a lightweight

complete correctly.

that must coordinate with each other in order to

A user's job consists of multiple processes/threads.
Clustered computers

- The old Cedar system and current RISC processor
- The old Cedar system
- The old Cedar system

proceeding with processing

of systems (alone with vector processing within a system)

they are often used simultaneously on certain types

multiple processors

programming are the two main approaches on

shared memory and distributed memory

paradigm and code constructs used

paradigm and code constructs used

code the user provides depends on the programming

the function of the process relative to the high-level

-
private process variables are supported in software in order to guarantee correctness must be specified at some level of coding.

- control(scheduling)/synchronization is explicit, i.e., deciding when and where operations are performed in
- communication between processes/threads is implicit, i.e., via load and stores to shared memory
- all processes/threads share the same address space, i.e., if process 1 and process 2 both ask for the contents of address 15 they get the same information.
code

tasks are usually controlled more explicitly by user
interactions via runtime system support, larger grain
processes/threads on each processor that consume

the use of loops typically implies worker

parallel constructs range from loops to large grain

•

•
the data be sent and received

computing process so synchronization follows from
required data is present in the private memory of the
operations can only be performed when all of the
control (scheduling) synchronization is implicit, i.e.

messages are sent and received by processes

communication between processes is explicit, i.e.

(typically in their private physical memory bank
and each of which is in their private address space (and
address if they access two different physical locations
process 1 and process 2 both ask for the contents of
each process has a distinct address space, i.e. if

Distributed Memory
based on its process number

each processor and that determines what it is to do

typically user code specifies a task that is sent to

(global data structures or global mathematical

complicate things since users often think in terms of

shared variables do not exist in any way (this can •

objects)