

---

---

# PetaOps Operating System Scalability

---

# Prologue

---

- My comments are predicated on the assumption that the operating system needs to be, at a minimum, “Unix-like”
- The PetaOps system must fit into the existing computing universe
  - system APIs
  - networking
  - required middleware
  - remote code development
  - existing code base
  - etc.

---

# What about NT?

---

This space intentionally left blank.

---

## — Apology (not)

- At past PetaOps workshops I have sounded cynical or “sour-grapes”
- Today is no exception
- In defense, I view it as being pragmatic

---

# A reference PetaOps system

---

- If we can get the HTMT processors at 100 Ghz (say 200 Gflops)
  - 5,000 processors
- If we are forced to go with commodity processors at, say, 8 Gflops each
  - 125,000 processors

---

# Concurrency

---

- For latency hiding we need somewhere between 1 million and 100 million threads
- Let's stop for just a moment and think about this....

Okay now lets get started again

---

# Quiz

---

- Question:
  - What is the most difficult application on the planet to parallelize?
- Answer:
  - Unix

---

# Sanity check

---

- Where are we today?
- The largest SSI Unix system is 256 processors
- We are a factor of 20–500 off the mark
- From first-hand experience
  - every factor of 2 increase in o/s scalability induces at least a factor of 10 of effort
- Why is it that SGI and HP's architectures support  $N$  processor ccNUMA and the o/s is at  $N/x$ , where  $x > 1$ ?
  - Because it's hard to do otherwise!

---

# What is Unix?

---

- A **single processor** operating system that has been stretched to handle SMPs
- The fundamental structure of the Unix internals precludes it's scalability, without complete overhaul, to thousands of processors
- Two significant areas of concern
  - the process manager (PM)
  - the virtual memory manager (VM)
- Too many critical sections

---

# — What are the major o/s issues?

---

- The amount of shared information in the internal structures
  - the `proc` structure is especially nasty
    - ▲ it has been a catch-basin for years
- The need to maintain a single-system image
- Aside: the internal data structures have not changed all that much since the Thompson and Ritchie days

---

# Miscellaneous

---

- Unix LOVES linked-lists
  - Think about walking a linked-list of PetaOps scale and comparing against a single field in a structure
- Maintaining consistency in VM is particularly troubling
  - many memory levels, separate page pools, consistency between “nodes”, etc.
- Data movement within the o/s
  - buffer cache, for example

---

## o/s–architecture induction

---

- Should the architecture influence (dictate?) the operating system?
- Should the operating system force architectural decisions?
- This is not a rhetorical question

---

# Availability

---

- Sheer component count of a PetaOps system dictates frequent failures in all types of components
  - By frequent I mean minutes
- The operating system must be resilient to failures of
  - disks (easy :-)
  - processors
  - memory
  - interconnects
  - ASICs
- This has a **profound** effect on the o/s

---

## Availability (cont'd)

---

- Speaking from first-hand experience doing a robust, in the face of failures, o/s is a very difficult problem
- To not have this capability in the o/s for a PetaOps system is a recipe for certain failure
  - failure here means a system that is always either booting or doing application start-up

---

# What parallel programming model?

- A slight digression that is related to the o/s
- What programming model do we want or need?
  - shared-memory
  - distributed-memory
- More on this later

---

# Shared-memory

---

- Assume we are going to go with a single address space PetaOps system
  - We are in serious trouble here
- I have no genuine feelings of possible success here unless the o/s is “completely” restructured
- We do have some data on which we can extrapolate
  - It is not encouraging

---

## Shared-memory (cont'd)

---

- Targeting 2007 for availability
- A relatively small team could immediately begin the redesign and define the internals
- Architectural simulators will be required far in advance of the actual hardware
- As the specifics of the machine become available the machdep work could begin

---

## Distributed o/s

---

- This benign sounding approach is not trivial
- Availability is still a significant issue
- The scale of the operating systems's domains are more manageable, say  $\approx$ (500) processors
  - Unix as it exists today might suffice
- This implies a message-passing programming model

---

# Programming methodology

---

- If its distributed, it is MPI
  - grep “MPI” with “message-passing” in what follows
- Lets consider the consequences of MPI, at the PetaOPs scale, on the operating system
- In what follows I am not “picking on” MPI
  - it is a vehicle to point out the operating system issues that surface at Peta scales

---

# MPI issues

---

- Consider process start-up and tear-down
  - If we have a  $M \times N$  system ( $M$  nodes of  $N$  processors) we can get  $M$ -way parallelism in start-up and tear-down
  - For the  $N$  processors we are, with the usual Unix semantics, stuck with linearity in the PM
  - At the concurrency scale we need this is a significant overhead
- Recommendation:
  - Relax the usual Unix semantics and allow, for example, a “gang fork” that is atomic to mitigate the Unix imposed semantics’ overhead

---

## MPI issues (cont'd)

---

- Consider attaching memory to the  $N$  processes for fast messaging
  - This goes through the VM and it is linear

---

## MPI issues (cont'd)

---

- Communication between processes
  - daemon or  $n^2$  connections?
- How long does it take to open  $M \times N$  sockets?
  - Can we have  $10^6$  sockets?
- How much of a bottleneck is a single, or few, daemon(s)?
  - Fast start-up and tear-down with a daemon but sub-optimal runtime performance

---

## — MPI issues (cont'd)

- How do we do gang scheduling across a million distributed processes?

---

## Recommendation

---

- The operating system for the PetaOps system must be “Unix”
  - The internals will be different
- A single shared-memory operating system approach may well be a intractable problem

---

# Conclusions

---

- It is my opinion that nothing about a PetaOps system or its intended applications is easy or even well understood
- I believe the operating system may well be the most difficult aspect
- Work on the operating system needs to be initiated, and sustained, starting now
  - It needs to have the target architecture(s) in view
- A single team is the most (only?) viable approach