Parallel Processing Using Linux PCs: What Is And Isn't There

Hank Dietz
Associate Professor of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285
hankd@ecn.purdue.edu
http://dynamo.ecn.purdue.edu/~hankd

What Is And Isn't There?

- **Parallel PC architectures:**
  - **SIMD Within A Register (e.g. Intel MMX)**
  - **Shared-memory multiprocessors (e.g., Intel MPS)**
  - **Attached processors**
  - **Clusters (networks?)**
- **The Linux OS:**
  (`http://sunsite.unc.edu/mdw/linus/html`**)**
  - **A nice, free, highly popular UNIX**
  - **Support for parallelprocessing?**

SIMD Within A Register (SWAR)

- Popularized as multimedia support
- Partition 32-bit/64-bit/128-bit registers, datapaths, and function units into multiple k-bit fields
- Perform SIMD operations across fields
- Integer operations only
- Improved badwidth, Loads/Stores treat fields as a block
- RISC-like SIMD control minimizes VLSI complexity, pipeline constraints

SWAR Hardware Architecture

- **AMD K6 MMX (MultiMedia eXtensions)**
- **Cyrix M12 MMX (MultiMedia eXtensions)**
- **Digital Alpha MAX (MultimediA eXtensions)**
- **Hewlett-Packard PA-RISC MAX (Multimedia Acceleration eXtensions)**
- **Intel Pentium & Pentium 2 MMX (MultiMedia eXtensions)**
- **Sun SPARC V9 VIS (Visual Instruction Set)**
- **MIPS Digital Media eXtension (MDMX, "Mad Max")**
- **PowerPC will have SWAR support**
- ***Ordinary 32-bit/64-bit processors***

A General SWAR Model

- Manufacturer SWAR support is machine dependent
  - Different (often irregular) instructions
  - Different width registers, fields
  - Different register use constraints
    (e.g., can't mix MMX with floating point)
  - HLL models specify each instruction
- Need complete SIMD/vector features
- Need variable size/parallelism-width data
- Cannot have HLL-visible "holes"
  (i.e., omit quirky SWAR instructions)

How to fill the SWAR Gaps?

- A simple example ...
- Use ordinary 32-bit/64-bit op, but correct for field interactions
- For 8 4-bit fields in a 32-bit register:

```
return (x + y);
t = ( ( x & 0x77777777) + (y & 0x77777777));
return (t ^ ((x § y) & 0x88888888));
```

SWAR Summary & Status

- **Promising "new" integer SIMD/vector technology**
  - **+ SIMD, tiny latency, cheap communication**
  - **– Doesn't do MIMD nor message-passing**
- **Linux (or other OS) doesn't care**
- **Developing SWAR module languages/compilers**
  `http://dynamo.ecn.purdue.edu/~hankd/SWAR/`

Shared -Memory Multiprocessors

- Multiple processors share some address space
- Processors communicate by
  - Storing into and loading from shared space
  - Signals or interrupts
- Access time not constant, but a function of:
  - Address (e.g., local versus remote)
  - History (e.g., caching, ownership protocols)

Shared Memory Architecture

- **Basic SMP software interface:**
  - **Intel Multiprocessor Standard (MPS) 1.1 and 1.4,** `http://www.intel.com/IAL/processr/mpovr.htm`
  - **Does not specify hardware implementation nor performance**
- **Shared cache, shared bus, multiple busses, etc.**
- **Many vendors ... cheap for 2-4 PEs** `http://www.uruk.org/~erich/mps-hw.html`

Linux Support for SMP

- **Linux 2.0 kernel *standarly* supports**
  – MPS with up to 16 Pentium Pro, Pentium, or 486
  – Sun4m SPARC machines (not our focus here)
- **Must rebuild kernel ...**
  only change is uncomment `SMP=1` in makefile
- **Currently only 1 CPU in kernel at a time**
  + Everything works
  – Locked system calls, dumb scheduler

Programming Models

- **"Shared everything"**
  (Gnu C libraries are not thread-safe)
  - `bb_threads` **uses Linux** `clone()` **call**
- **"Shared something"**
  - **System V IPC shared memory segments**
  - `mmap()` **segments**
- `volatile`, **"false sharing," scheduler issues, etc.**
  `http:/yara.ecn.purdue.edu/~pplinus/ppsmp.html`
  + **MIMD, µs latency, shared-memory, message-passing**
  - **Surprises, scaling trouble, I/O contention**

Attached Processors

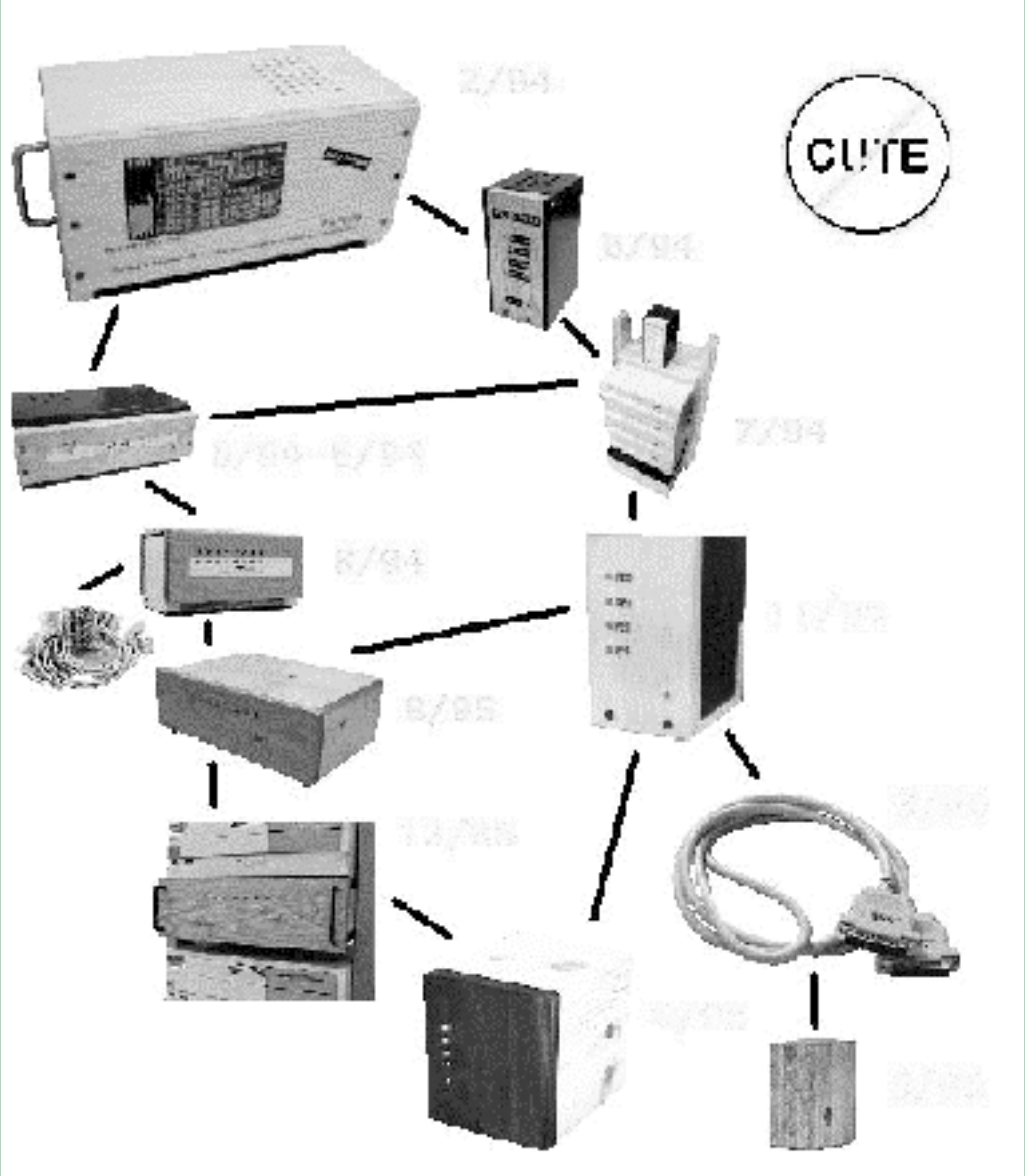
- Linux PC hosts other processors
- Lots of add-ons, all different
  (e.g., DSP cards, reconfigurable FPGAs, ICE DRE)
- Linux PCs are good hosts:
  - Full source code, “hacking” guides
  - Good near-real-time scheduling
  - Linux uses Intel I/O port protection
  - Can fun DOS tools under `dosemu`
- Very cost effective … once you’ve done it ;-)

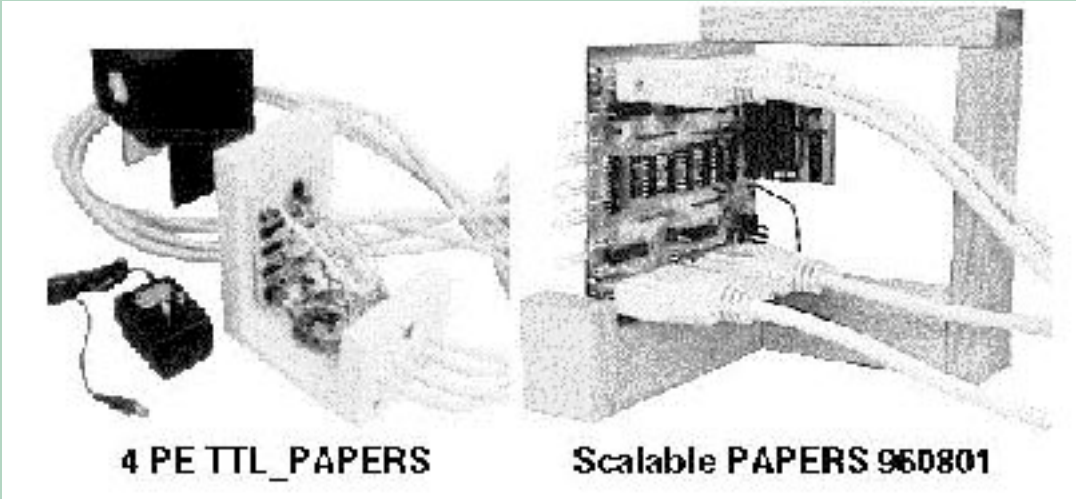
Clusters (Networks?)

- **Some networks do not scale:**
  CAPERS, ParaPC, PLIP, & SLIP
- **Some networks are not (yet) widely available:**
  SCI, FC, SHRIMP, ParaPC, USB, & Firewire
- **Some networks are not (yet) supported by Linux:**
  HiPPI, Serial HiPPI, SCI, FC, ParaPC, USB, & Firewire
- **This leaves only:**
  Myrinet, ParaStation, PAPERS, ATM, various Ethernets, & ARCNET

Use	Myrinet	ParaStation	PAPERS	ATM	Ethernet	ARCNET
MIMD	Good	Good	Fair	Good	Good	Good
SIMD	Poor	Fair	Good	Poor	Poor	Poor
Message Passing	Good	Good	Poor	Good	Good	Good
Shared Memory	Fair	Fair	Good	Poor	Poor	Poor
Aggregate Functions	Poor	Poor	Good	Poor	Poor	Poor
Bandwidth	Good	Good	Poor	Good	Good	Poor
Latency	Good	Good	Good	Fair	Poor	Poor
Standard?	No	No	No	Yes	Yes	Yes
PC Cost Multiplier	1.9x	2.0x	1.1x	2.5x	1.1x 1.3x sw	1.2x

Assumes PC Cost is $2,000 ...


CUTE



- **PAPERS is Purdue's Adapter for Parallel Execution and Rapid Synchronization**
- **Aggregate function network for parallel processing**
- **Public hardware design and AFAPI library**

What's In PAPERS?

- Each PC connects via standard parallel port ...
- Will improve bandwidth, add aggregates
- PAPERS unit contains 4 subsystems:
  - Fast barrier synchronization hardware
  - LED status display
  - Aggregate function data communication
  - Parallel signal support

Barrier Synchronization

- An *arbitrary group* of PCs can participate
- No PC executes past a barrier until *all* have arrived
- Two-cycle barriers using two barrier units:
  1. Output: present at barrier, reset barrier
  2. Input: poll for *all* PCs present
- Synchronization time:
  - < 0.1 μs logic, ≤0.2 μs cable, 1 μs port register
  - Total <3 μs for 4 PCs; <7 μs for 2,000+ PCs

LED Status Display

- Blue power on
- Bi-color LED/PE
  - Green: running
  - Red: at barrier
  - Orange: in OS
  - Black: not parallel code

Aggregate Functions

- **Not shared memory nor message passing: a *group* of PEs initiates each op**
- **As a barrier side-effect, each PC can:**
  1. Output: a datum
  2. Input: selected *function of data from all PCs*
- **Can sample global state in O(1) time**
- **Examples: broadcast, multi-broadcast, associative reductions, scans (parallel prefix), SIMD/VLIW conditionals, voting and scheduling ops**

Performance (μs latency)

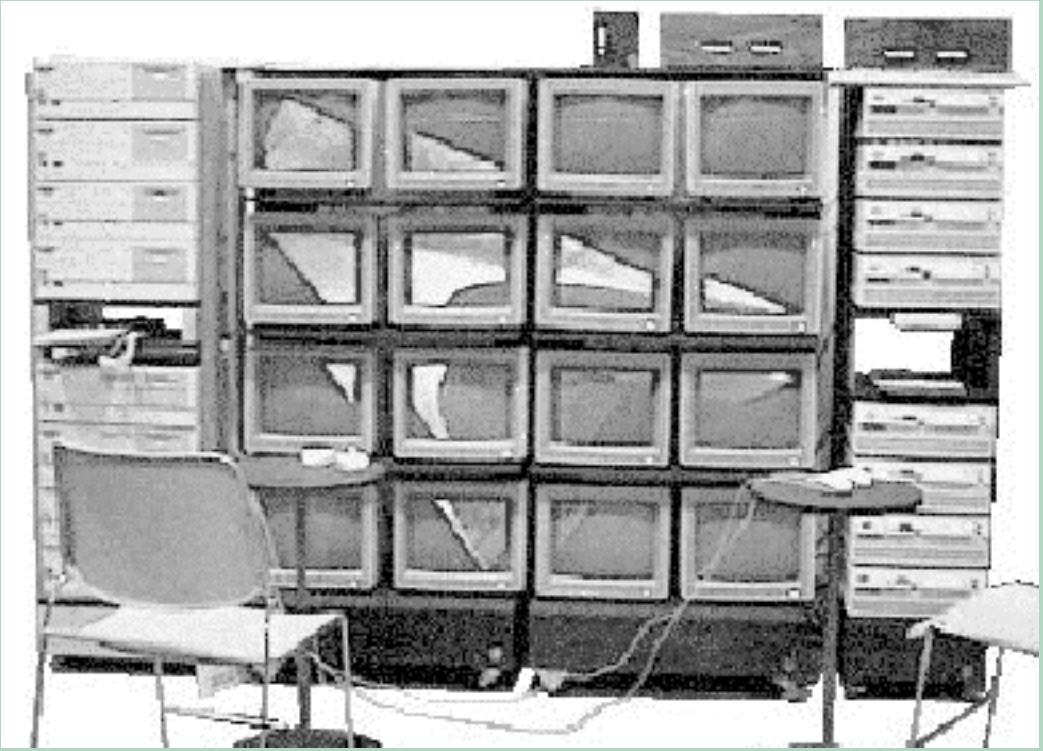
Machine	Barrier Sync.	32-bit PutGet (permutation)	64-bit Broadcast
MasPar MP-1	0.1	44.0	31.0
486 Linux (PAPERS1)	3.1	27.0	81.0
Cray T3D (PVM)	21.0		82.0
486 Linux (TTL_PAPERS)	2.5	216.0	137.0
Intel Paragon XP/S	530.0	700.0	210.0
486 Linux (Ethernet PVM3)	49,000.0	100,000.0	40,000.0

Parallel Signals

- Any PC can asynchronously signal *all* PCs
- Separate barrier used for signal acknowledge
- Used for:
  - Parallel job control, gang scheduling
  - Asynchronous shared memory
  - Asynchronous message passing

A Simple Demonstration

- Cluster of 4 Linux sub-notebooks
- Multi-voice music
- PCs agree on who plays each note;
  PCs without notes have red LEDs
- Note played by toggling PC speaker
- Without precise coordination, it isn't music



- **Integer SWAR promising, but there's work to do**
- **SMP MPS useful now, but not as good as it seems**
- **Attached processors get an "M" rating**
- **Clusters with appropriate connections work well (e.g., 100 Mb/s Ethernet + PAPERS)**
- **"Stock" LANs work only for coarse-grain MIMD**