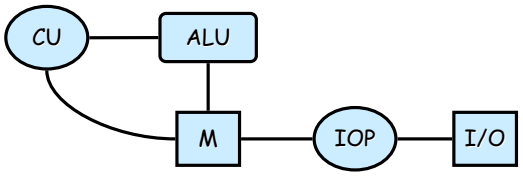
第一部分 并行计算机体系结构（模型

- 并行计算概念
- 并行计算互连模型
- 并行计算结构模型
- 并行计算性能评测

串行计算机中的并行性



- 中断机制使CPU与慢速外设之间重叠执行;
- IOP使得快速外设与CPU可以并行操作;
- 多任务系统中多个程序可以交替地使用单个CPU以提高系统吞吐率和CPU利用率;

Impediments to Parallel Computing

- Algorithm development is harder
  - —complexity of specifying and coordinating concurrent activities
- Software development is much harder
  - —lack of standardized & effective development tools, programming models, and environments
- Rapid pace of change in computer system architecture
  - —today's hot parallel algorithm may not be a good match for tomorrow's parallel computer!

串行计算机中的并行性（续）

取指令	I1	I2	I3	Jump			
取操作数	--	I1	I2	I3	Jump		
执行	--	--	I1	I2	I3	Jump	
存储结果	--	--	--	I1	I2	I3	Jump
时间	1	2	3	4	5	6	7

- 流水线技术可使取指令/执行周期中的步骤可以重叠执行;
- 多重算术部件;
- VLIW等

1.1 并行性概念

- **并行性**

把问题中具有可以同时进行运算或操作的特性称为并行性(parallelism). 开发并行性的目的是为了对问题进行并行处理, 以提高求解问题的效率

- **并行性的体现**

同时性(simultaneity)指两个或多个事件在同一时刻发生;

并发性(concurrency)指两个或多个事件在同一时间间隔内发生

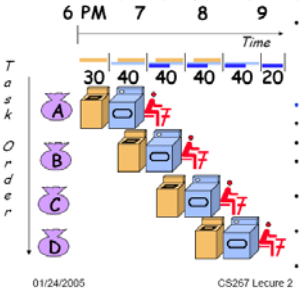
- **并行性等级**

指令内部; 指令之间;任务或进程级;作业或任务级

What is Pipelining?

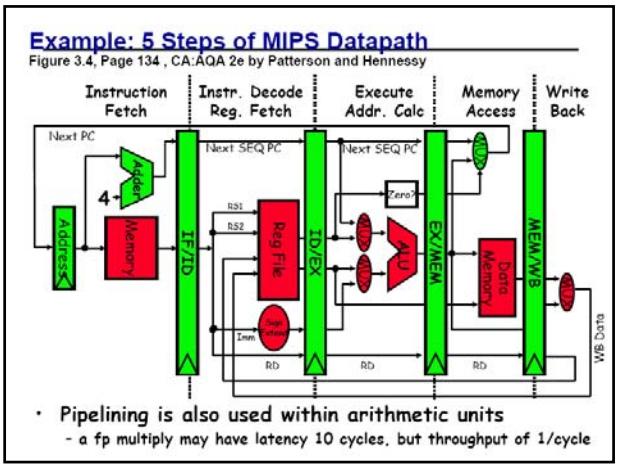
Dave Patterson's Laundry example: 4 people doing laundry

wash (30 min) + dry (40 min) + fold (20 min) = 90 min Latency



- In this example:
  - Sequential execution takes 4 * 90min = 6 hours
  - Pipelined execution takes 30+4*40+20 = 3.5 hours
- Bandwidth = loads/hour
- BW = 4/6 l/h w/o pipelining
- BW = 4/3.5 l/h w pipelining
- BW <= 1.5 l/h w pipelining, more total loads
- Pipelining helps bandwidth but not latency (90 min)
- Bandwidth limited by slowest pipeline stage
- Potential speedup = Number pipe stages

01/24/2005 CS267 Lecure 2 6



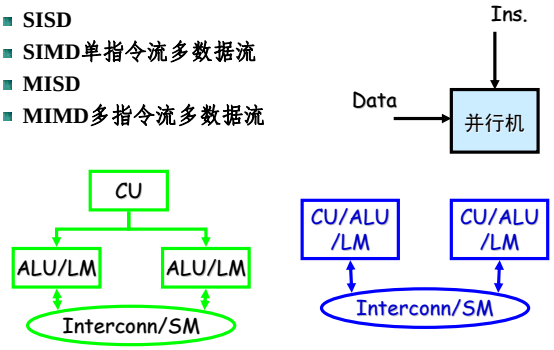
- 并行性的开发途径
  - 资源共享: 利用软件的方法让多个用户按一定时间顺序轮流地使用同一套资源,以提高其利用率, 可相应提高系统性能.
    - 网络打印机
    - 多道程序、分时OS →真正的处理机代替虚拟机→分布处理系统

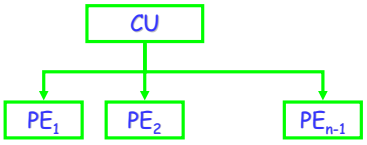
1.1并行性概念（续）

- 并行性的开发途径
  - 时间重叠: 在并行性概念中引入时间因素,让多个处理过程在时间上相互错开,轮流重叠地使用同一套硬件设备的各个部分,以加快硬件周转而赢得速度. 如流水线.
    - 分离、细化功能部件→流水线→功能不同的多机系统→异构型多处理机系统
  - 资源重复: 在并行性概念中引入空间因素,通过重复设置硬件资源来提高可靠性或性能. 如双机系统.
    - 多操作部件和多体存储器→相联、并行处理机→同构型多处理机系统

1.2并行计算机分类(Flynn)

- SISD
- SIMD单指令流多数据流
- MISD
- MIMD多指令流多数据流





1.3并行计算的研究方向

- **并行机体系结构**从并行计算的角度主要研究高性能计算机系统的体系结构与存储模型、高速互连网络、通信操作、多级存储及其一致性等;
- **并行算法设计**重点研究并行算法的常用设计策略、基本设计技术、一般设计过程、标准性能评测等;
- **并行程序设计**主要研究并行程序设计模型、共享存储和分布存储系统的并行编程、并行程序的设计环境与工具以及科学计算可视化等。
- **并行应用** 集中在应用领域并行算法的研究上，包括计算生物学、计算化学、计算流体动力学、飞行动力学、计算机辅助设计、数据库管理、油藏建模、中长期天气预报、海洋环流和求解N-body问题等以及面向应用的大型科学与工程问题的并行数值计算，诸如求解大型稀疏方程组、大型非线性方程和有限元分析等。

1.3并行计算的研究方向（续）

- **非传统计算** 在交叉学科中出现了一些非传统计算方式，例如分子计算和量子计算等。这些计算本身包含着巨大的并行度，超出了传统的计算模式。分子计算中参与计算的分子数可达1020，我们可以利用大量分子的并行操作能力，以空间换时间的方式提高计算能力，从而可望解决常规计算方式难以解决的问题。

并行计算机性能指标常用术语

- **Megaflops 10^6 flops**
- **Gigaflops 10^9 flops workstations**
- **Teraflops 10^{12} top 17 supercomputers**
- **by 2005 every supercomputer in the top 500?**
- **Petaflops 10^{15} 2010?**

The Need for Speed: Complex Problems

- Science
  - —understanding matter from elementary particles to cosmology
  - —storm forecasting and climate prediction
  - —understanding biochemical processes of living organisms
- Engineering
  - —combustion and engine design
  - —computational fluid dynamics and airplane design
  - —earthquake and structural modeling
  - —pollution modeling and remediation planning
  - —molecular nanotechnology
- Business
  - —computational finance
  - —information retrieval
  - —data mining
- Defense
  - —nuclear weapons stewardship
  - —cryptology

高效能计算系统计划（HPCS）的进展

- 美国国防部高级研究计划局（DARPA）于2001 年初在高性能计算领域提出了HPCS 计划，2002 年中开始实施。该计划面向千万亿（peta）规模计算机系统的需求，针对当前高端计算机系统开发及应用中存在的问题，研发适应高端国家安全应用的高性能计算系统，以填补目前基于80 年代末技术的高性能计算和未来的量子计算之间的空白。

1.4并行计算机应用开发计划

- **并行计算：并行机上所作的计算，又称高性能计算或超级计算。**
- **计算科学：计算物理、计算化学、计算生物等**
- **科学与工程问题的需求：气象预报、油藏模拟、核武器数值模拟、航天器设计、基因测序等。**
- **需求类型：计算密集、数据密集、网络密集。**
- **美国HPCC计划：重大挑战性课题，3T性能**
- **美国DARPA的HPCS（Petaflops）研究计划。 Pflop/s**
- **美国能源部ASCI计划：核武器数值模拟。**

已完成的**ASCI**系统

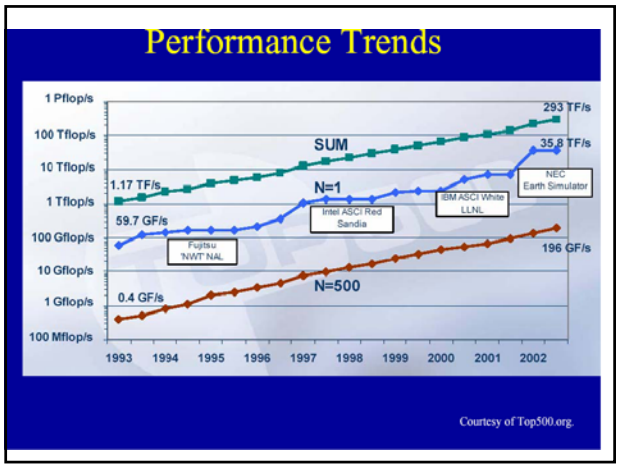
名称	处理器数	峰值(TF)	完成时间
ASCI红色（Intel）	9632	03.207	1998年
ASCI蓝山（SGI）	6144	03.072	1999年
ASCI蓝洋（IBM）	5808	03.868	2000年
ASCI白色（IBM）	8192	12.288	2001年
ASCI Q （HP）	9632	30	2004年

正在研制的**ASCI**系统

名称	处理器数	峰值（TF）	计划完成时间
ASCI紫色 （IBM）	12544	100	2004
Blue Gene/L（IBM）	13000	360	2005
Red Storm （Cray）	10368	40~100	2004

Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	131072	2005	280600	367000
2	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40960	2005	91290	114688
3	DOE/NNSA/LLNL United States	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	12208	2006	75760	92781
4	NASA/Ames Research Center/NAS United States	Columbia - SGI Altix 1.5 GHz, Voltaire Infiniband SGI	10160	2004	51870	60960
5	Commissariat a l'Energie Atomique (CEA) France	Tera-10 - NovaScale 5160, Itanium2 1.6 GHz, QuadricsBull SA	8704	2006	42900	55705.6
6	Sandia National Laboratories United States	Thunderbird - PowerEdge 1850, 3.6 GHz, InfinibandDell	9024	2006	38270	64972.8
7	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME Grid Cluster - Sun Fire X64 Cluster, Opteron 2.4/2.6 GHz, Infiniband NEC/Sun	10368	2006	38180	49868.8
8	Forschungszentrum Juelich (FZJ) Germany	JUBL - eServer Blue Gene Solution IBM	16384	2006	37330	45875
9	Sandia National Laboratories United States	Red Storm Cray XT3, 2.0 GHz Cray Inc.	10880	2005	36190	43520
10	The Earth Simulator Center Japan	Earth-Simulator NEC	5120	2002	35860	40960

- Rmax **Maximal LINPACK performance achieved**
- Rpeak **Theoretical peak performance**
- LINPACK是一种进行浮点性能测试的标准。
- Nmax **Problem size for achieving Rmax**
- N1/2 **Problem size for achieving half of Rmax**

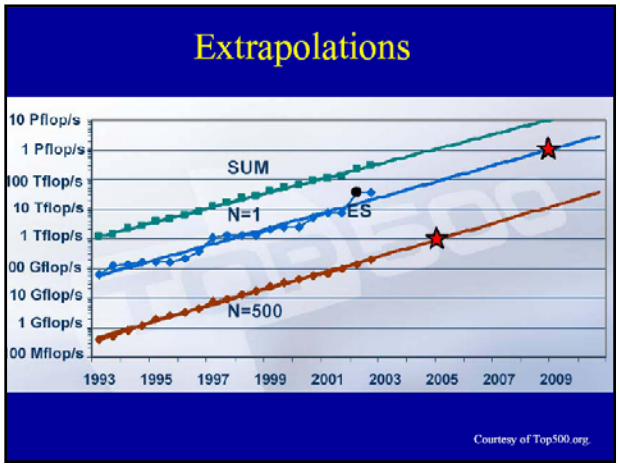


Scale of Today's Largest HPC Systems

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz , Voltaire Infiniband / 2008 IBM	129600	1105.00	1456.70	2483.47
2	Oak Ridge National Laboratory United States	Jaguar - Cray XT5 QC 2.3 GHz / 2008 Cray Inc.	150152	1059.00	1381.40	6950.60
3	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz / 2008 SGI	51200	487.01	608.83	2090.00
4	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
5	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	450.30	557.06	1260.00
6	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband / 2008 Sun Microsystems	62976	433.20	579.38	2000.00
7	NERSC/LBNL United States	Franklin - Cray XT4 QuadCore 2.3 GHz / 2008 Cray Inc.	38642	266.30	355.51	1150.00
8	Oak Ridge National Laboratory United States	Jaguar - Cray XT4 QuadCore 2.1 GHz / 2008 Cray Inc.	30976	205.00	260.20	1580.71
9	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, XT3/4, 2.4/2.2 GHz dual/quad core / 2008 Cray Inc.	38208	204.20	284.00	2506.00
10	Shanghai Supercomputer Center China	Dawning 5000A - Dawning 5000A, QC Opteron 1.9 Ghz, Infiniband, Windows HPC 2008 / 2008 Dawning	30720	180.60	233.47	

>1 petaflop

> 100K cores

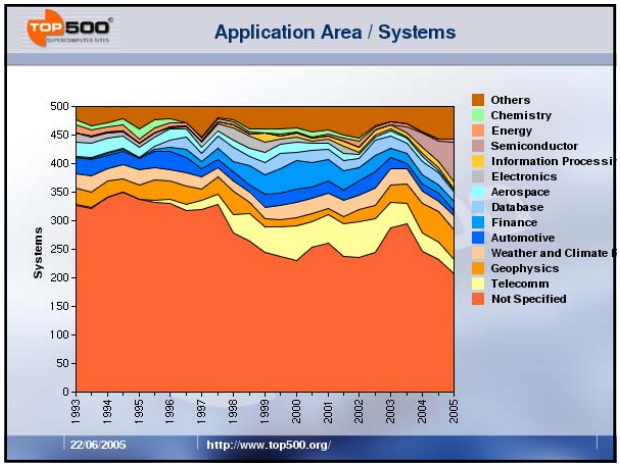


The Dawn of Petascale Systems

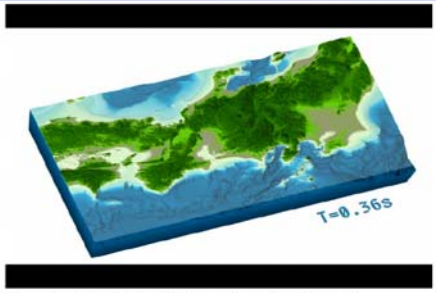
(PetaFLOPS = 10^{15} FLoating-point Operations Per Second)

- LANL RoadRunner
  - —hybrid architecture:
    - 6,562 dual-core AMD Opteron processors
    - 12,240 Cell B.E. processors
  - —Infiniband interconnect
  - —peak performance ~1.5 petaflop
- ORNL Jaguar system
  - —quad-core 2.3GHz AMD Barcelona processors
  - —over 181K processor cores
  - —toroidal interconnect topology: Cray Seastar2+
  - —peak performance ~1.4 petaflop
  - —deployed fall 2008

Image credits: http://www.lanl.gov/news/albums/computer/Roadrunner_1207.jpg

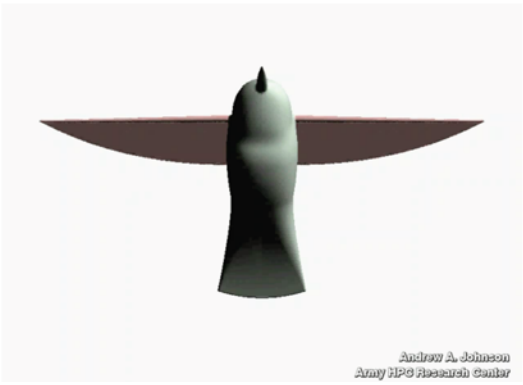


Earthquake Simulation

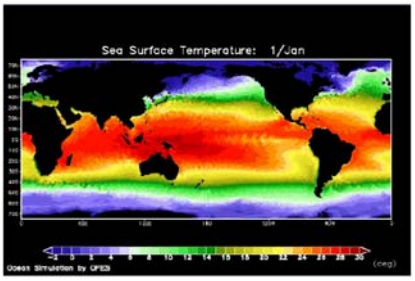


Earthquake Research Institute, University of Tokyo
Tonankai-Tokai Earthquake Scenario
Photo Credit: The Earth Simulator Art Gallery, CD-ROM, March 2004

Air Velocity (Front)

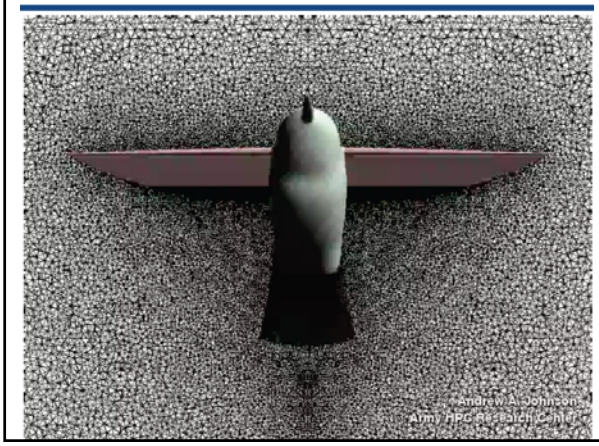


Ocean Simulation



Ocean Global Circulation Model for the Earth Simulator
Seasonal Variation of Ocean Temperature
Photo Credit: The Earth Simulator Art Gallery, CD-ROM, March 2004

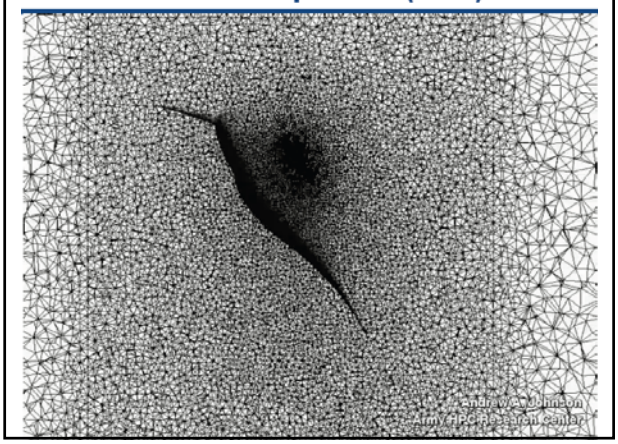
Mesh Adaptation (front)



Fluid-Structure Interactions

- Simulate …
  - —rotational geometries (e.g. engines, pumps), flapping wings
- Traditionally, such simulations use a fixed mesh
  - —drawback: solution quality is only as good as initial mesh
- Dynamic mesh computational fluid dynamics
  - —integrate automatic mesh generation within parallel flow solver
    - nodes added in response to user-specified refinement criteria
    - nodes deleted when no longer needed
    - element connectivity changes to maintain Delaunay mesh
  - —mesh changes continuously as geometry + solution changes
- Sophisticated parallelization in Unified Parallel C
- Example: 3D simulation of a hummingbird's flight

Mesh Adaptation (side)



NSF Petascale Acquisition

Feb 2007: "Leadership-Class System Acquisition - Creating a Petascale Computing Environment for Science and Engineering"

- NSF's goal for 2006-2011
  "enable petascale science and engineering through the deployment and support of a world-class HPC environment comprising the most capable combination of HPC assets available to the academic community"
- Scientific aim: support computationally challenging problems
  —simulations that are intrinsically multi-scale, or
  —simulations involving interaction of multiple processes

Multi-Stage Switching Network

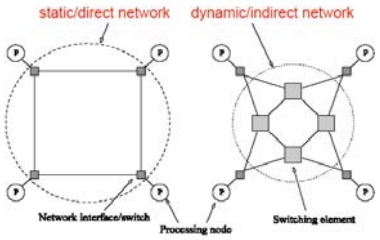
- **The Multi-Stage Switching Network is a dynamically configurable network. This means that the switches connecting the various processors can be changed dynamically, depending on where the messages need to be passed. The network presented here has 4 processors which are connected to 4 other processors through a 2 stage network with two-function switches. Crossbar switches "direct the traffic" in the network. If two processors are trying to send messages to two other processors in parallel, then sometimes a message may not get through if the crossbar switch is set straight and the receiving processor requires a switched position to connect to the sending processor. (See the Definitions section for more detailed explanation of the switching network).**

2. 并行计算结构模型

- 互联网络模型
- SIMD和MIMD结构模型
- 存储模型
- 编程举例

2.1 互联网络模型

- 网络的特征和术语
- 网络的拓扑结构
- 网络拓扑结构的连接函数表示



- **TOPOLOGY**
  The processors of the machine and the circuits in a switching network are arranged using a particular layout. Because not all nodes (i.e. processors) can have links to all other links in parallel, some pattern is used to decided how the pairs of nodes will be linked. This pattern usually follows some general rule and is called the topology **of the network.**

Interconnection Networks

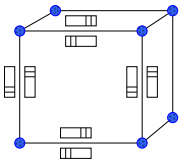
- Switch: maps a fixed number of inputs to outputs
  - Number of ports on a switch = *degree* of the switch.
  - Switch cost
    - grows as the square of switch degree
    - peripheral hardware grows linearly with switch degree
    - packaging cost grows linearly with the number of pins
- Key property: blocking vs. non-blocking
  - blocking
    - path from p to q may conflict with path from r to s
    - for independent p, q, r, s
  - non-blocking
    - disjoint paths between each pair of independent sources and sinks

Network Interface : Processor node's link to the interconnect

- Network interface responsibilities
  - packetizing communication data
  - computing routing information
  - buffering incoming/outgoing data
- Network interface locations
  - I/O bus: PCI or PCIx on many modern systems
  - memory bus: e.g. Opteron HyperTransport
    - higher bandwidth and tighter coupling than I/O bus
- Network performance
  - depends on relative speeds of I/O and memory buses

Historical Perspective

- Early machines were:
  - Collection of microprocessors.
  - Communication was performed using bi-directional queues between nearest neighbors.
- Messages were forwarded by processors on path.
  - "Store and forward" networking
- There was a strong emphasis on topology in algorithms, in order to minimize the number of hops.



Network Analogy

- To have a large number of transfers occurring at once, you need a large number of distinct wires.
- Networks are like streets:
  - Link = street.
  - Switch = intersection.
  - Distances (hops) = number of blocks traveled.
  - Routing algorithm = travel plan.
- Properties:
  - Latency: how long to get between nodes in the network.
  - Bandwidth: how much data can be moved per unit time.
    - Bandwidth is limited by the number of wires and the rate at which each wire can accept data.

Characteristics of a Network

- Topology (how things are connected)
  - Crossbar, ring, 2-D and 2-D torus, hypercube, omega network.
- Routing algorithm:
  - Example: all east-west then all north-south (avoids deadlock).
- Switching strategy:
  - Circuit switching: full path reserved for entire message, like the telephone.
  - Packet switching: message broken into separately-routed packets, like the post office.
- Flow control (what if there is congestion):
  - Stall, store data temporarily in buffers, re-route data to other nodes, tell source node to temporarily halt, discard, etc.

Properties of a Network: Latency

- Diameter: the maximum (over all pairs of nodes) of the shortest path between a given pair of nodes.
- Latency: delay between send and receive times
  - Latency tends to vary widely across architectures
  - Vendors often report hardware latencies (wire time)
  - Application programmers care about software latencies (user program to user program)
- Observations:
  - Hardware/software latencies often differ by 1-2 orders of magnitude
  - Maximum hardware latency varies with diameter, but the variation in software latency is usually negligible
- Latency is important for programs with many small messages

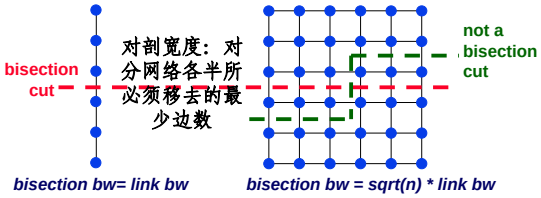
硬件时延与软件时延

- 发送：
  - **应用程序把要发送的数据拷贝到OS的buf中**
  - **OS根据要发送的数据计算出检查和，并把它加在消息中，同时启动超时计数器**
  - **OS把buf中的数据送NIC并通知硬件开始发送消息**
- 接收：
  - **系统把数据从NIC硬件拷贝到OS的buf中**
  - **根据收到的数据计算出检查和，如果与发送的匹配则接收方发一个回答信号给发送方，否则删除这个消息，因为发送方在超时后会重发这个消息**
  - **若通过检查和则系统把收到的数据拷贝到用户地址空间中并启动应用程序继续**

- 带宽bandwidth：消息进入网络后，网路传输信息的最大速率。Mb/s
- 传输时间transmission time：消息通过网络（中一点）的时间，等于消息长度除以频宽
- 飞行时间：消息第一位到达接收方所花费的时间，包括由于网络中转发或其他硬件引起的时延
- 传输时延transport latency ＝飞行时间＋传输时间
- 发送方开销sender overhead：处理器把消息放到互联网络的时间，包括软硬件花费的时间
- 接收方开销
- 总时延＝发送方开销＋传输时延＋接收方开销

Properties of a Network: Bisection Bandwidth:

- Bisection bandwidth: bandwidth across smallest cut that divides network into two equal halves
- Bandwidth across "narrowest" part of the network



- Bisection bandwidth is important for algorithms in which all processors need to communicate with all others

●例：带宽10Mb/s，发送方开销230μs，接收方270 μs，2台机器相距100米，发送1000字节消息给另一台机器

$$\text{总时延} = 230\mu s + \frac{0.1km}{0.5\times 2997925km/s} + \frac{1000\times 8b}{10Mb/s} + 270\mu s$$
$$= 1031\mu s$$

●相距1000公里

$$\text{总时延} = 230\mu s + \frac{1000\times 10^6}{0.5\times 299792.5} + \frac{1000\times 8}{10} + 270\mu s$$
$$= 7971\mu s$$

网络的特征（续）

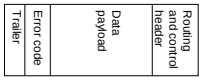
- 节点度（Node Degree）：射入或射出一个节点的边数。在单向网络中，入射和出射边之和称为节点度。
- 对剖带宽（ Bisection Bandwidth）:每秒钟内，在最小的对剖平面上通过所有连线的最大信息位（或字节）数
- 如果从任一节点观看网络都一样，则称网络为对称的（Symmetry）

Properties of a Network: Bandwidth

- A network is partitioned into two or more disjoint sub-graphs if some nodes cannot reach others.
- The bandwidth of a link = w * 1/t
  - w is the number of wires
  - t is the time per bit

Unidirectional: in one direction
Bidirectional: in both directions

- Bandwidth typically in Gigabytes (GB), i.e., 8* 2^{20} bits
- Effective bandwidth is usually lower than physical link bandwidth due to packet overhead.



网络带宽：网路传输信息的最大速率Mb/s

- Bandwidth is important for applications with mostly large messages

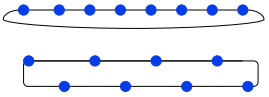
静态互连网络 与动态互连网络

- 静态互连网络：处理单元间有着固定连接的一类网络，在程序执行期间，这种点到点的链接保持不变；典型的静态网络有一维线性阵列、二维网孔、树连接、超立方网络、立方环、洗牌交换网、蝶形网络等
- 动态网络：用交换开关构成的，可按应用程序的要求动态地改变连接组态；典型的动态网络包括总线、交叉开关和多级互连网络等。

Linear and Ring Topologies

- Linear array

  - Diameter = n-1; average distance ~n/3.
  - Bisection bandwidth = 1 (units are link bandwidth).
- Torus or Ring

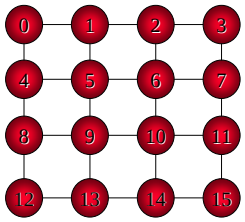


  - Diameter = n/2; average distance ~ n/4.
  - Bisection bandwidth = 2.
  - Natural for algorithms that work with 1D arrays.

静态互联网络的连接函数

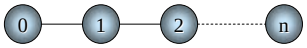
✧2-D网孔

$$MC^2_{-1}(p) = p-1, \quad 0 \le p \le n-1 \wedge pMOD\sqrt{n} \ne 0$$
$$MC^2_{+1}(p) = p+1, \quad 0 \le p \le n-1 \wedge (p+1)MOD\sqrt{n} \ne 0$$
$$MC^2_{-\sqrt{n}}(p) = p-\sqrt{n}, \quad \sqrt{n} \le p \le n-1$$
$$MC^2_{+\sqrt{n}}(p) = p+\sqrt{n}, \quad 0 \le p \le n-\sqrt{n}$$



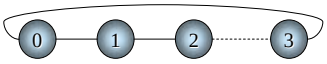
静态互联网络的连接函数

✧一维线性阵列



$$LC_{-1}(p) = p-1, \quad 1 \le p \le n-1$$
$$LC_{+1}(p) = p+1, \quad 0 \le p \le n-2$$

✧环

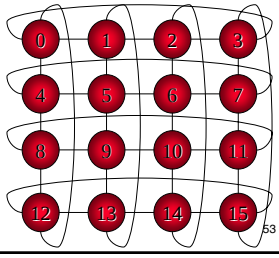


$$RC_{-1}(p) = (n+p-1)MODn, \quad 0 \le p < n$$
$$RC_{+1}(p) = (p+1)MODn, \quad 0 \le p < n$$

静态互联网络的连接函数

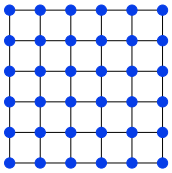
$$MC^2_{-1}(p) = \left((p-1)DIV\sqrt{n}\right) \times \sqrt{n} + \left(p-1+\sqrt{n}\right)MOD\sqrt{n}, \quad 0 \le p \le n-1$$
$$MC^2_{+1}(p) = \left((p-1)DIV\sqrt{n}\right) \times \sqrt{n} + \left(p-1+\sqrt{n}\right)MOD\sqrt{n}, \quad 0 \le p \le n-1$$
$$MC^2_{-\sqrt{n}}(p) = \left(n+p-\sqrt{n}\right)MODn, \quad 0 \le p \le n-1$$
$$MC^2_{+\sqrt{n}}(p) = \left(n+p+\sqrt{n}\right)MODn, \quad 0 \le p \le n-1$$

✧2-D环绕

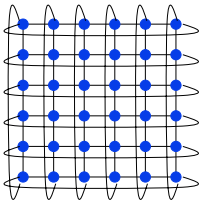


Meshes and Torus

Two dimensional mesh
- **Diameter = 2 * (sqrt(n) – 1)**
- **Bisection bandwidth = sqrt(n)**



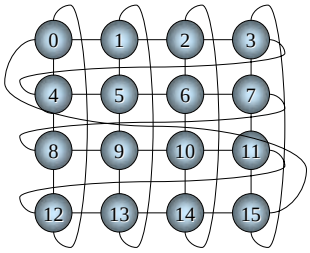
Two dimensional torus
- **Diameter = sqrt(n)**
- **Bisection bandwidth = 2* sqrt(n)**



- **Generalizes to higher dimensions (Cray T3D used 3D Torus).**
- **Natural for algorithms that work with 2D and/or 3D arrays.**

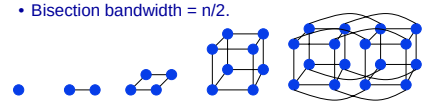
静态互联网络的连接函数

✧2-D Illiac



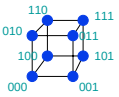
Hypercubes

- Number of nodes n = 2^d for dimension d.
  - Diameter = d.
  - Bisection bandwidth = n/2.



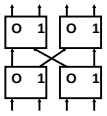
- 0d 1d 2d 3d 4d

- Popular in early machines (Intel iPSC, NCUBE).
  - Lots of clever algorithms.
- Gray code addressing:
  - Each node connected to d others with 1 bit different.

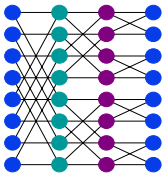


Butterflies

- Diameter = log n.
- Bisection bandwidth = n.
- Cost: lots of wires.
- Used in BBN Butterfly.
- Natural for FFT.



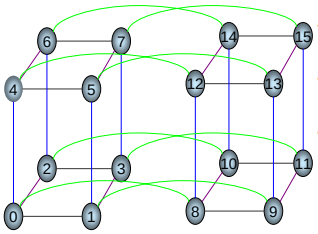
butterfly switch



multistage butterfly network

n=2^m个结点的超立方体的连接函数

$$CC_i(p_{m-1}\cdots p_{i+1}p_i p_{i-1}\cdots p_0) = p_{m-1}\cdots p_{i+1}\overline{p_i}p_{i-1}\cdots p_0, \quad 0 \le i < m$$

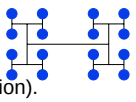


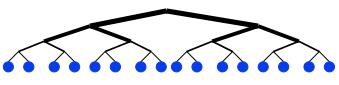
- **Distance between any two nodes is at most *log n*.**
- **Each node has *log n* neighbors**
- **Distance between two nodes = # of bit positions that differ between node numbers**

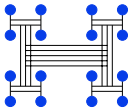
网络	直径	对剖宽度	弧连通性	成本
星型	2	1	1	$p-1$
线性阵列	$p-1$	1	1	$p-1$
环	$\lfloor p/2 \rfloor$	2	2	p
2D网孔	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2D环绕	$2\lfloor \sqrt{p}/2 \rfloor$	$2\sqrt{p}$	4	$2p$
Illiac网孔	$\sqrt{p}-1$	$2\sqrt{p}$	4	$2p$
超立方体	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
k-d环绕	$d\lfloor k/2 \rfloor$	$2k^{d-1}$	$2d$	dp
二叉树	$2(\lceil \log p \rceil - 1)$	1	1	$p-1$
全连接	1	$p^2/4$	$p-1$	$p(p-1)/2$

Trees

- Diameter = log n.
- Bisection bandwidth = 1.
- Easy layout as planar graph.
- Many tree algorithms (e.g., summation).
- Fat trees avoid bisection bandwidth problem:
  - More (or wider) links near top.
  - Example: Thinking Machines CM-5.







Topologies in Real Machines

older — newer

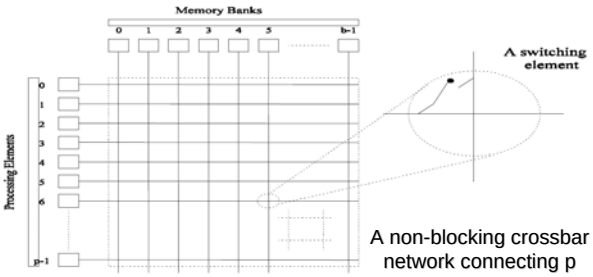
Red Storm (Opteron + Cray network, future)	3D Mesh
Blue Gene/L	3D Torus
SGI Altix	Fat tree
Cray X1	4D Hypercube*
Myricom (Millennium)	Arbitrary
Quadrics (in HP Alpha server clusters)	Fat tree
IBM SP	Fat tree (approx)
SGI Origin	Hypercube
Intel Paragon (old)	2D Mesh
BBN Butterfly (really old)	Butterfly

Many of these are approximations: E.g., the X1 is really a "quad bristled hypercube" and some of the fat tree are not as fat as they should be at the top

动态互联网络-BUS

- **All processors access a common bus for exchanging data**
- **Used in simplest and earliest parallel machines**
- **Advantages**
  - **distance between any two nodes is *O(1)***
  - **provides a convenient broadcast media**
- **Disadvantages**
  - **bus bandwidth is a performance bottleneck**
  - **bus-based machines are typically limited to dozens of nodes**

A crossbar network uses an $p \times m$ grid of switches to connect p inputs to m outputs in a non-blocking manner

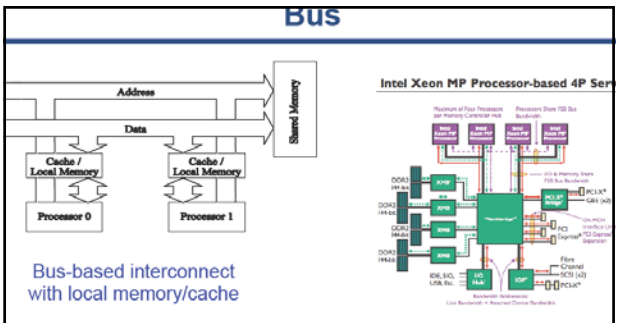


A non-blocking crossbar network connecting p processors to b memory banks

- Since much of the data accessed by processors is local to the processor, a local memory can improve the performance of bus-based machines
- Assuming that each processor accesses k data items, and each data access takes time t_{cycle}, the execution time is lower bounded by $t_{cycle}*kp$ seconds.
- Let us assume that 50% of the memory accesses are made to local data. $0.5*t_{cycle}*k+0.5*t_{cycle}*kp$

Crossbar Network

- **Cost of a crossbar: $O(p^2)$, $p=m$**
- **Scalable in terms of performance but unscalable in terms of cost**
- **Examples**
  - **Earth Simulator: custom 640-way single-stage crossbar**
  - **RTC: Myrinet 2000 interconnect**
    - **16-way crossbar switches in 128-way Clos network**

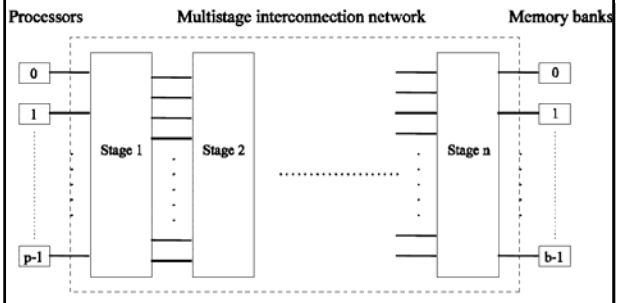


Since much of the data accessed by processors is local to the processor, a local memory can improve the performance of bus-based machines

Assessing Network Alternatives

- **Buses**
  - **excellent cost scalability**
  - **poor performance scalability**
- **Crossbars**
  - **excellent performance scalability**
  - **poor cost scalability**
- **Multistage interconnects**
  - **compromise between these extremes**

Multistage Network



Schematic of processor-to-memory multistage interconnection network (e.g., BBN Monarch)

Omega Network Switches

- **2 × 2 switches connect perfect shuffles**
- **Each switch operates in two modes**

Pass-through

Cross-over

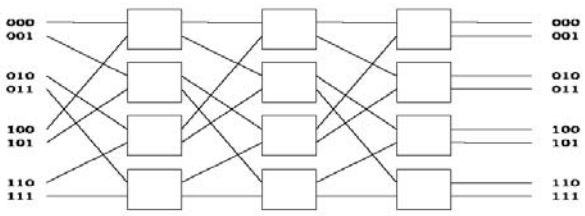
Multistage Omega Network

- **Organization**
  - ***log p* stages**
  - ***p* inputs/outputs**
- **At each stage, input *i* is connected to output *j* if:**

$$j = \begin{cases} 2i, & 0 \le i \le p/2 - 1 \\ 2i + 1 - p, & p/2 \le i \le p - 1 \end{cases}$$

if $p = 2^k$ then j = left_rotate(i)

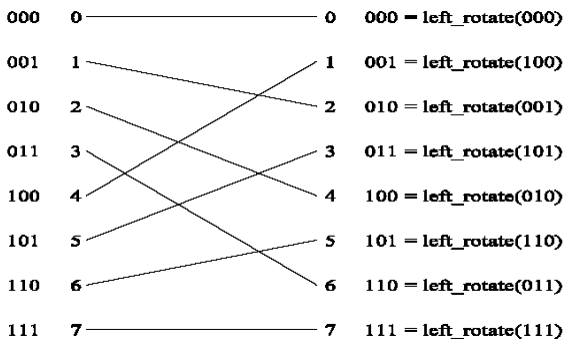
Multistage Omega Network



Omega network connecting eight inputs and eight outputs

Cost: p/2 × log p switching nodes → O(*p log p*)

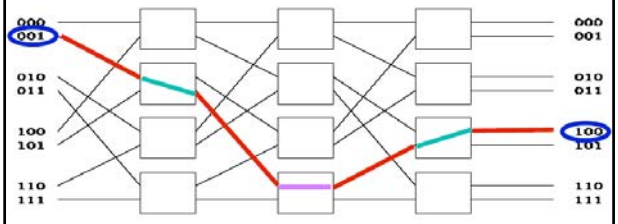
Each Omega stage is connected in a perfect shuffle



Omega Network Routing

- **Let**
  - ***s* = binary representation of the source processor**
  - ***d* = binary representation of the destination processor or memory**
- **The data traverses the link to the first switching node**
  - **IF the most significant bit of *s* and *d* are the same**
  - **THEN route data in pass-through mode by the switch**
  - **ELSE use crossover path**
- **Strip off leftmost bit of s and d**
- **Repeat for each of the *log p* switching stages**

Omega Network Routing



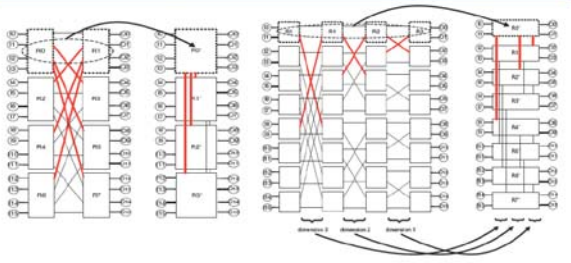
Example: s= 001 → d=100

stage 1: leftmost bit s != d → crossover

stage 2: middle bit == d → pass-through

stage 3: rightmost bit s != d → crossover

Butterfly and Flattened Butterfly



4-ary, 2 fly → 4-ary, 2-flat 2-ary, 4 fly → 2-ary, 4-flat

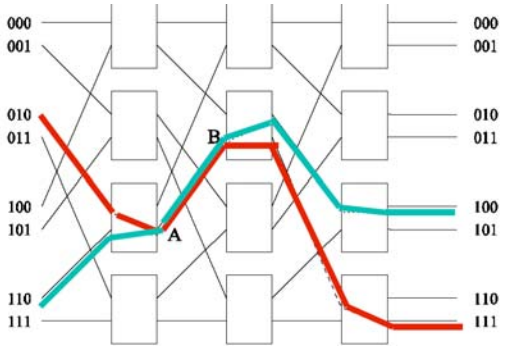
- Start with conventional butterfly k-ary n-fly
- Flatten routers in each row of the network into single router
- Flattened butterfly has better performance and path diversity

John Kim, William J. Dally, Dennis Abts: Flattened butterfly: a cost-efficient topology for high-radix networks. ISCA 2007: 126-137

22

Blocking in an Omega Network

One of the messages (010 to 111 or 110 to 100) blocks at link AB



嵌入

- Mapping a graph G(V,E) into G'(V',E')
  - Congestion = maximum # edges in E mapped onto 1 edge in E'
  - Dilation = maximum # edges in E' mapped onto 1 edge in E
  - Expansion = (# nodes in V')/(# nodes in V)
- **如果各系数为1，则称为完美嵌入。**
  - **环网可完美嵌入到2-D环绕网中**
  - **超立方网可完美嵌入到2-D环绕网中**

Metrics for Dynamic Interconnection Networks

Network	Diameter	Bisection Width	Cost
Crossbar	1	*p*	*p*^2
Omega Network	log *p*	*p*/2	*p*log*p*

Cost: **~ # links or switches**

Binary Reflected GRAY code:

G(*i*,*d*) denotes the *i*-th entry in a sequence of Gray codes of *d* bits. G(*i*,*d*+1) is derived from G(*i*,*d*) by reflecting the table and prefixing the reflected entry with 1 and the original entry with 0.

G(i,x):

G(0,1)=0, G(1,1)=1,

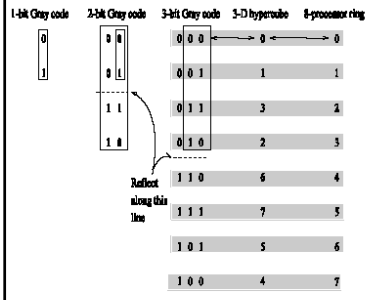
G(i,x+1)=IF i<2^x THEN G(i,x) ELSE 2^x+G(2^{x+1}-1-i,x)

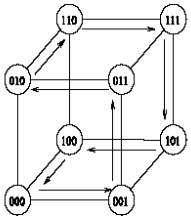
Example of BRG code

1-bit	2-bit	3-bit	8p ring	8p hyper
0	0 0	0 0 0	0	0
1	0 1	0 0 1	1	1
	1 1	0 1 1	2	3
	1 0	0 1 0	3	2
		1 1 0	4	6
		1 1 1	5	7
		1 0 1	6	5
		1 0 0	7	4

24

Embedding a Linear Array into a Hypercube





3-bit reflected Gray code ring

embedding into a 3D hypercube

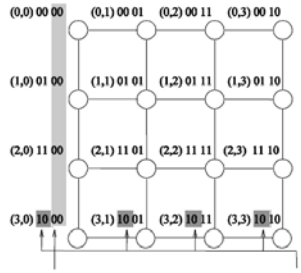
Embedding other networks on hypercubes:

Hypercube is a rich topology, many other networks can be "easily" mapped onto it.

- Mapping a linear array into an hypercube:
  A linear array (or ring) of 2^d processors can be embedded into a d-dimensional hypercube by mapping processor I onto processor G(I,d) of the hypercube
- Mapping a 2^r x 2^s mesh on an hypercube:
  - processor(i,j)---> G(i,r)||G(j,s) (|| denote concatenation)

25

Embedding a Mesh into a Hypercube



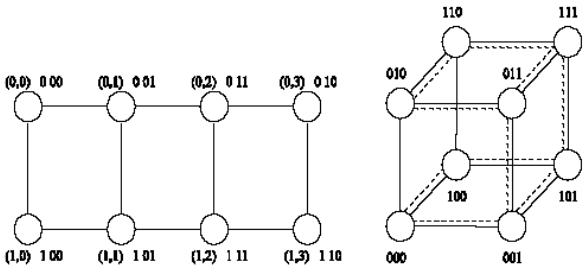
Mapping nodes in 4 x 4 mesh to nodes in a 4D hypercube

Congestion, dilation, and expansion of the mapping is 1

Embedding a Linear Array into a Hypercube

- **Given**
  - **linear array (or ring) of 2^d nodes**
  - ***d*-dimensional hypercube**
- **Map**
  - **node *i* of the linear array → node *G(i, d)* of the hypercube**

Embedding a Mesh into a Hypercube



Embedding a 2 × 4 mesh into a 3D hypercube

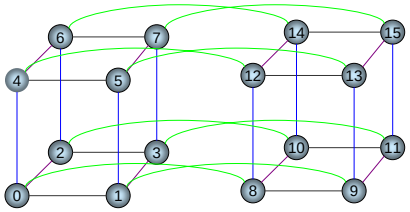
Congestion, dilation, and expansion of the mapping is 1

嵌入： 参考文献

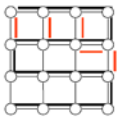
- **LIU Fang'ai, LIU Zhiyong, ZHANG Yongsheng, The embedding of rings and meshes into RP(k)networks. SCIENCE IN CHINA SERIES F , 2004 Vol.47 No.5 P.669-680**

n=2^m个结点的超立方体上n 个数求和

$$CC_i(p_{m-1}\cdots p_{i+1}p_ip_{i-1}\cdots p_0)=p_{m-1}\cdots p_{i+1}\overline{p_i}p_{i-1}\cdots p_0,\quad 0\le i<m$$

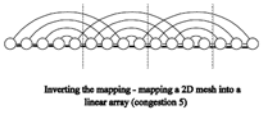


Embedding a Mesh into a Linear Array



Mapping a linear array into a 2D mesh (congestion 1).

Embedding a 16 node linear array into a 2-D mesh



Inverting the mapping - mapping a 2D mesh into a linear array (congestion 5)

Inverse of the mapping: 2D mesh to 16-node linear array

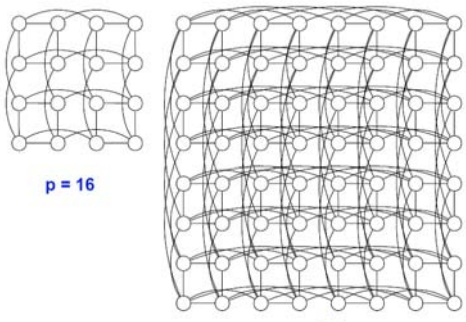
Key:

Dark lines: links in the linear array
Normal lines: links in the mesh.

37

开始		CC_0	CC_1	CC_2	CC_3	备注
0	a	a+b	a+b+ c+d	+abcdefhg	+a..p	16个数求和，初始时假定数据a..p已经放到对应的结点中（网络直径）。然后依次按照CC0、CC1、CC2、CC3、交换数据后相加，最终结果在结点0中。实际上每个结点中的结果一样，只是求和的次序不一样。
1	b	b+a	+badc	+badc fehg		
2	c	c+d	c+d+ a+b	?		
3	d	d+c	+dcba	?		
4	e	e+f	e+f+ g+h	+efghabcd		
5	f	f+e	+fehg	?		
6	g	g+h	g+h+ e+f	?		
7	h	h+g	+hgfe	?		
8	i	i+j	i+j+ k+l	+ijklmnop		
9	j	j+i	+jilk	?		
10	k	k+l	k+l+ i+j	?		
11	l	l+k	+lkji	?		
12	m	m+n	m+n+ o+p	+mnopijkl		
13	n	n+m	+nmpo	?		
14	o	o+p	o+p+ m+n	?		
15	p	p+o	+ponm	?		

Embedding a Hypercube into a 2-D Mesh



p = 16

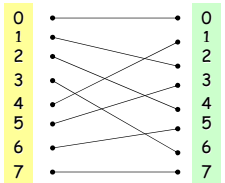
p = 64

39

洗牌交换网络

$$SHuffle(p_{m-1}p_{m-2}\cdots p_1p_0)=p_{m-2}p_{m-3}\cdots p_0p_{m-1}$$

$$EXchange(p_{m-1}p_{m-2}\cdots p_1p_0)=p_{m-1}p_{m-2}\cdots p_1\overline{p_0}$$



开始		+(SH,EX)		+(SH,EX)		+(SH,EX)	+(SH,EX)	备注
0	a	(a,i)	ai	(ai,)	aiem	(aiem,)	(aiemckgo,)	16个数求和，初始时假定数据a..p与结点对应．进行$\log_2 n$次+(SH，EX)操作．任一结点经过SH得到一个数，再经过EX得到一个数，二者相加．最终结果在结点0中．
1	b	(i,a)	ia	(em,)	emai	(ckgo,)	(bjfndllhp,)	
2	c	(b,j)	bj	(ia,)	iame	(,)		
3	d	(j,b)	jb	(me,)	meia	(,)		
4	e	(c,k)	ck	(bj,)	bjfn	(,)		
5	f	(k,c)	kc	(fn,)	fnbj	(,)		
6	g	(d,l)	dl	(jb,)	jbnf	(,)		
7	h	(l,d)	ld	(nf,)	nfjb	(,)		
8	i	(e,m)	em	(ck,)	ckgo	(bjfn,)		
9	j	(m,e)	me	(go,)	gock	(dlhp,)		
10	k	(f,n)	fn	(kc,)	kcog	(,)		
11	l	(n,f)	nf	(og,)	ogkc	(,)		
12	m	(g,o)	go	(dl,)	dlhp	(,)		
13	n	(o,g)	og	(hp,)	hpdl	(,)		
14	o	(h,p)	hp	(ld,)	ldph	(,)		
15	p	(p,h)	ph	(ph,)	phld	(,)		

Store-and-Forward Routing

- Message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop
- The total communication cost for a message of size m words to traverse l communication links is

$$t_{comm} = t_s + (mt_w + t_h)l$$

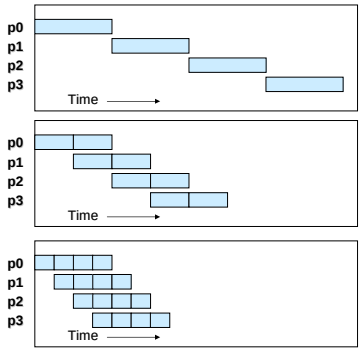
- In most platforms, t_h is small and the above expression
- can be approximated by

$$t_{comm} = t_s + mlt_w$$

例: SIMD-SE模型上的求和算法, n=2^m

```
begin
   for i=1 to logn do
      for all Pj where 0<=j<n do
      shuffle(aj)
      bj<-aj
      exchange(bj)
      aj<-aj+bj
      endfor
   endfor
end
```

Packet Routing : Store-and-forward makes poor use of communication resources



A single message sent over a store-and-forward network

The same message broken into two parts and sent over the network

The same message broken into four parts and sent over the network

Message Passing Costs

- Transfer time has three components
- *Startup time* (t_s)
  - time spent at sending and receiving nodes
    - executing the routing algorithm, programming routers, etc.
- *Per-hop time* (t_h)
  - includes factors such as switch latencies, network delays, etc.
- *Per-word transfer time* (t_w)
  - includes all overheads determined by the message length

Packet Routing

- Packet routing
  - breaks messages into packets
  - pipelines them through the network
- Packets may take different paths, thus each packet must carry
  - routing information
  - error checking
  - sequencing information
- t_s accounts for
  - Programming the network interfaces
  - Computing the routing information, etc.
- Static routing tables: all packets traverse the same path*

Packet Routing

- Packet size = r+s, r – original message, s – additional information carried in the packet
- The time for packetizing the message, mt_{w1}, is proportional to the length of the message.
- t_{w2} 传送每个字的时间（带宽的倒数）

$$t_{comm} = t_s + t_{w1}m + t_h l + t_{w2}(r+s) + (\frac{m}{r} - 1)t_{w2}(r+s)$$

$$= t_s + t_{w1}m + t_h l + t_{w2}m + t_{w2}\frac{s}{r}m$$

$$= t_s + t_h l + t_w m$$

$$t_w = t_{w1} + t_{w2}(1 + \frac{s}{r})$$

Cut-Through Routing

- Communication time for cut-through routing is

$$t_{comm} = t_s + mt_w + lt_h$$

- Identical to packet routing, but *tw* typically much smaller
- 如果*l*为1或消息很短则三种方式的一样
- 在link带宽一定的情况下，flit size的大小:
  - 小：片流速大，电路交换速度要快;
  - 大：buffer大，消息传输时延大;
  - 一般4bits ~ 32bytes
- Multilane cut-through routing

Packet Routing

- The total communication time for packet routing is

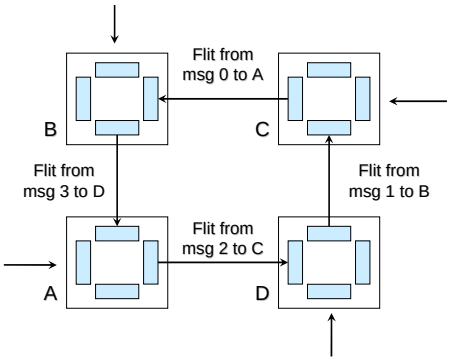
$$t_{comm} = t_s + t_w m + t_h l$$

- t_h可忽略时

$$t_{comm} = t_s + mt_w$$

- t_w accounts for overheads in packet headers

Deadlock in cut-through routing



Cut-Through Routing

- Takes concept of packet routing to an extreme
  - further divides messages into basic units called flits
  - Eliminate the overhead of transmitting routing information with each packet
- Flits are typically small → header information must be small
- For small headers
  - force all flits to take the same path, in sequence
    - a tracer is first sent to establish a connection
    - all flits then take same route
  - Associate error info at msg level instead of packet level.
  - Lean error detection mechanism can be used.

Simplified Cost Model for Cut-Through Routing

- The cost of communicating a message between two nodes *l* hops away using cut-through routing

$$t_{comm} = t_s + mt_w + lt_h$$

- Communicate in bulk
  - Aggregate small messages into a single large massage
  - On typical platforms such as clusters, t_s is much larger than t_h or t_w
- Minimize the volume of data
- Minimize distance of data transfer

Simplified Cost Model for Cut-Through Routing

- Typically, t_h smaller than t_s and t_w
  - t_s for smaller message
  - mt_w for lager message
- Often not possible to control routing and placement of tasks
- For these reasons, we approximate the cost of communication using cut-though routing transfer by

$$t_{comm} = t_s + mt_w$$

Simplified Cost Model for Messages

- Valid for only uncongested networks
- If a link takes multiple messages
  - corresponding t_w term must be scaled up by the # messages
- Network congestion varies by
  - communication pattern
  - match between pattern and network topology
- Communication models must account for congestion

References

- Based on Chapter 2 of "Introduction to Parallel Computing" by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003
- John Kim, William J. Dally, Dennis Abts: Flattened butterfly: a cost-efficient topology for high-radix networks. ISCA 2007:126-137.
- Lawrence C. Stewart and David Gingold. A New Generation of Cluster Interconnect . SiCortex White Paper. December 2006/revised April 2008.