Performance of Parallel Programs

Lecture Coverage

Performance Measures

- **Speed-up**
- **Scalability**
- **Isoefficiency**

Performance Issues

- **Computation Time**
- **Communication Time**
- **Wait Time**

Models and Formulas

- **Amdahl's Law**
- **Contention Free Models**
- **Operation counts and Asymptotic Analysis**

Problems

- **Load Balance**
- **Contention for Communication Resources**

Amdahl's Law

Amdahl's Law states that potential program speedup is defined by the fraction of code (P) which can be parallelized:

$$\text{speedup} = \frac{1}{1 - P}$$

If none of the code can be parallelized, P = 0 and the speedup = 1 (no speedup). If all of the code is parallelized, P = 1 and the speedup is infinite (in theory).

If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast. (Assuming an infinite number of processors and no communication or wait time.)

Amdahl's Law - Continued

Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

$$\text{speedup} = \frac{1}{P/N + S}$$

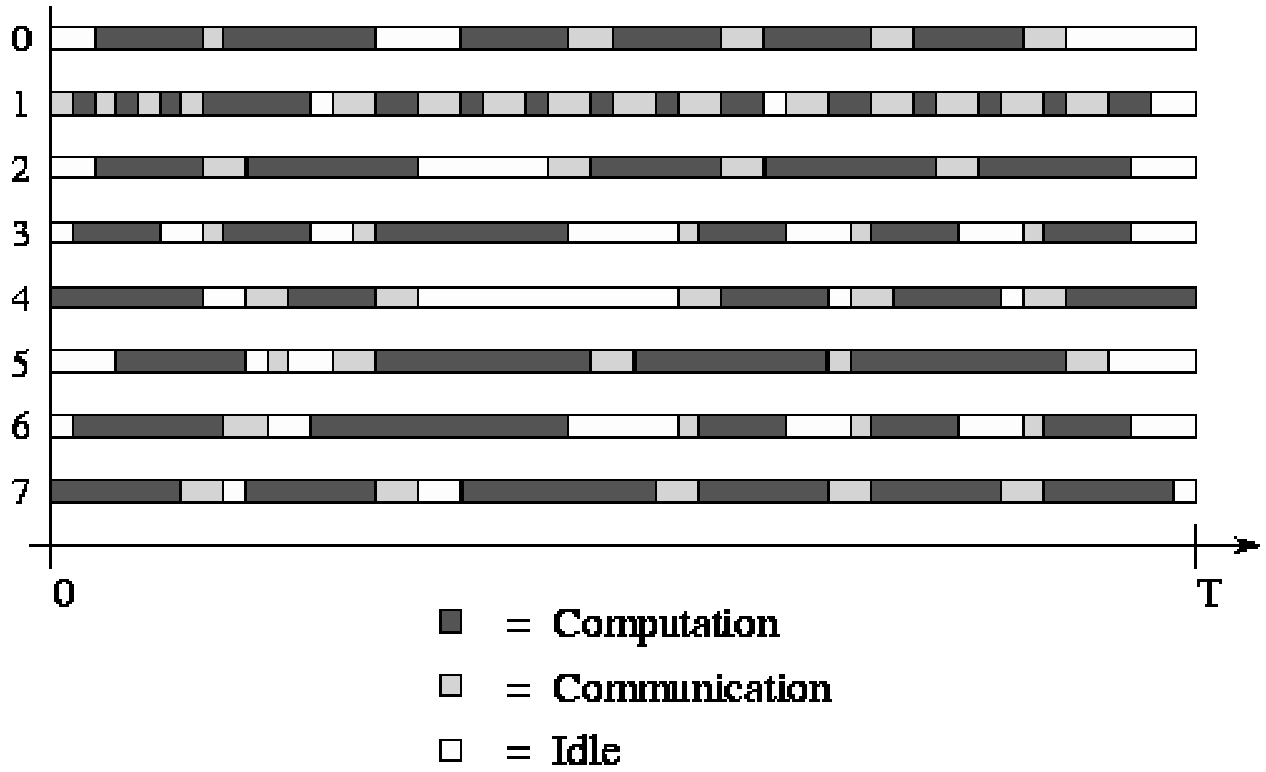
where P = parallel fraction, N = number of processors and S = serial fraction.

There are limits to the scalability of parallelism. For example, at P = .50, 0.90 and 0.99 (50%, 90% and 99% of the code is parallelizable):

	speedup		
N	P = .50	P = .90	P = .99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02

Simple Analytic Models - Comp+Comm+Idle

Execution behavior of a program



Simple Analytic Models

$$T = T\,(N,P,U,\text{-----})$$

$$T_j = \sum_{i=1}^{P} T^j_{comp} + \sum_{i=1}^{P} T^j_{comm} + \sum_{i=1}^{P} T^j_{idle}$$

$$T = (1/p)\,(\quad T^i_{comp} + \quad T^j_{comm} + \quad T^i_{idle})$$

or

$$T = \max\,(T^j_{comp} + T^j_{comm} + T^j_{idle})$$

$$= \max\,(T_j)$$

Simple Analytic Models - Continued

$T_{comp} = f(N_1, N_2 \text{ -- })$, N = count of some operations

$T_{comm} = \sum_i T^i_{msg}$,

$T_{msg} = t_s + t_w L$

t_s = latency or startup cost

t_w = transmission time/unit

L = length of message in units

T_{idle} = time waiting for communication or completion of other tasks

Estimation of Execution Time for Finite Grid Code

Consider a 3D mesh - NxNxZ

Let t_p = computation time for each grid point update

$T_{comp} = t_p N^2 Z$ - for single processor

For a 9 point stencil with 2xN points/processor

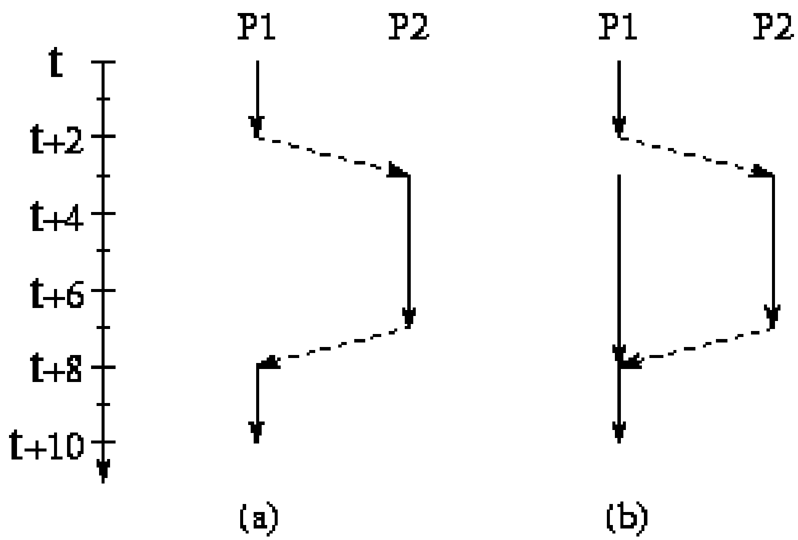
$L = 2NZ$ for each of two neighbors

$T_{comm} = 2P(t_s + t_w 2NZ)$

$T = (T_{comp} + T_{comm})/P$

$= t_c N^2 Z/P + 2t_s + 4(t_w NZ)$

Communication Time - Idle Time Trade-off



Efficiency and Speed-up

$$E_{relative} = T_1 / (P\ T_p)$$

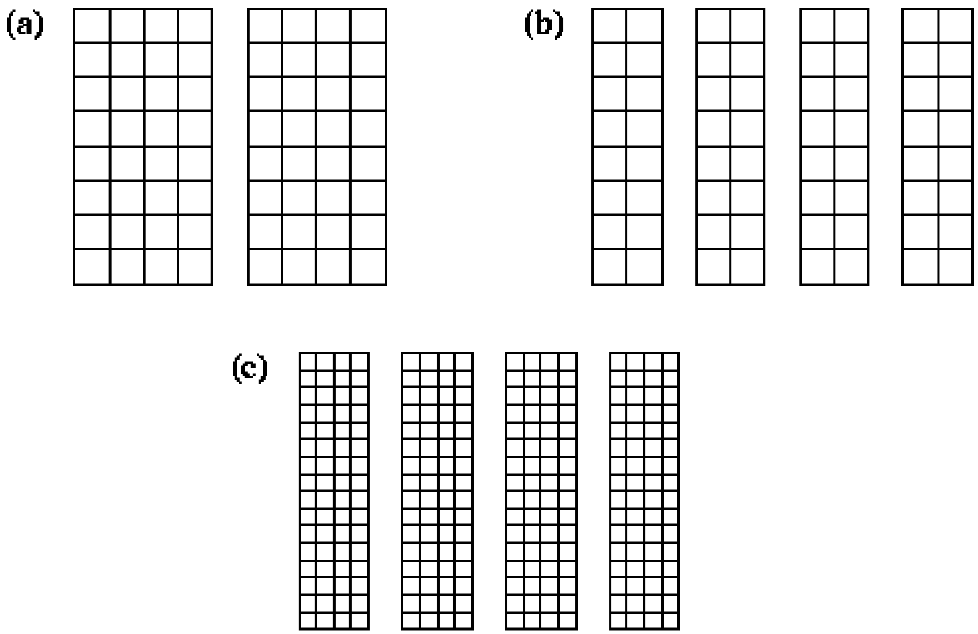
$$S_{relative} = P * E_1 = T_1 / T_p$$

$$E_{absolute} = T_1 \text{ (best sequential)} / (P\ T_p)$$

$$E_{relative} \text{ (grid comp.)} = t_p\ NxNxZ / (t_c N^2 t + 2 t_s\ P + 4NZPt_w)$$

$$(T_1 = t_c\ NxNxZ)$$

Parallel efficiency in partitioning of a 2-D grid.



Efficiency as a function of communication cost.

1-D partitioning of a 2-D grid

a) Partition among two processors. E decreases with respect to a)

b) Partition 128 points among four processors. E is the same as for a)

Scalability Analysis

What will be the speed-up or efficiency on P processors for N = M?

$$S = f(N,P),\ E = L(N,P)$$

What size problem can I reasonably solve on P processors?

$$T \propto g(N,P)$$

$$E = T_1 / (T_{comp} + T_{comm} + T_{idle})$$

For constant efficiency then T_1 must increase at the same rate as the parallel execution time.

Scalability Analysis

Isoefficiency metric - Establish a relationship between N and P such that E remains constant as P increases

- Call $T_{comm} + T_{idle} = T_{overhead}$

$T_{overhead} = T - T_{comm} - T_{idle} = T_O$

Isoefficiency metric

$T = (W + T_o\ (W, P))/P,\ W = T_1$

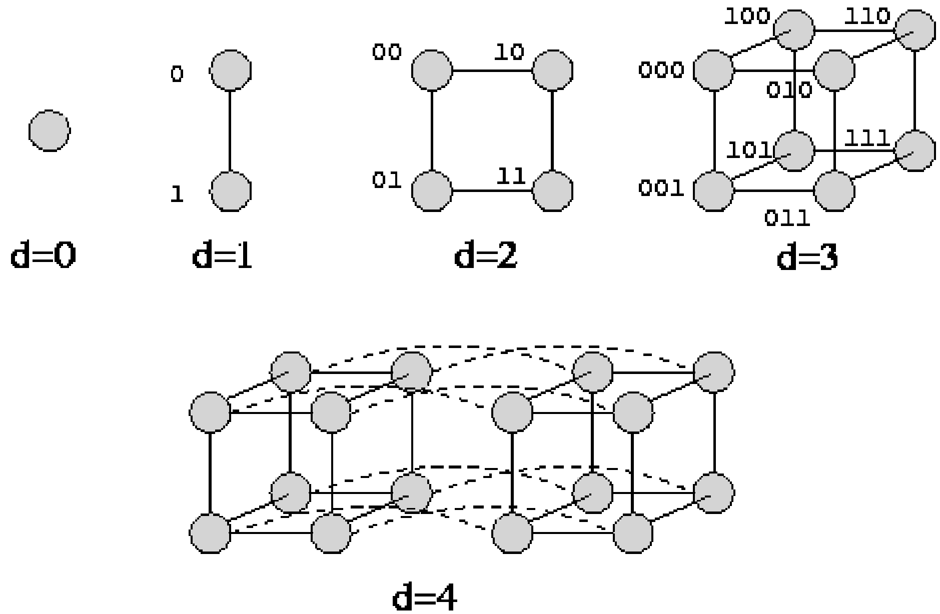
$S = W/T_p = WP/(W + T_o)$

$E = S/P = W/(W + T_o(W,P))$
$\quad = 1/(1 + T_o(W,P)/W)$

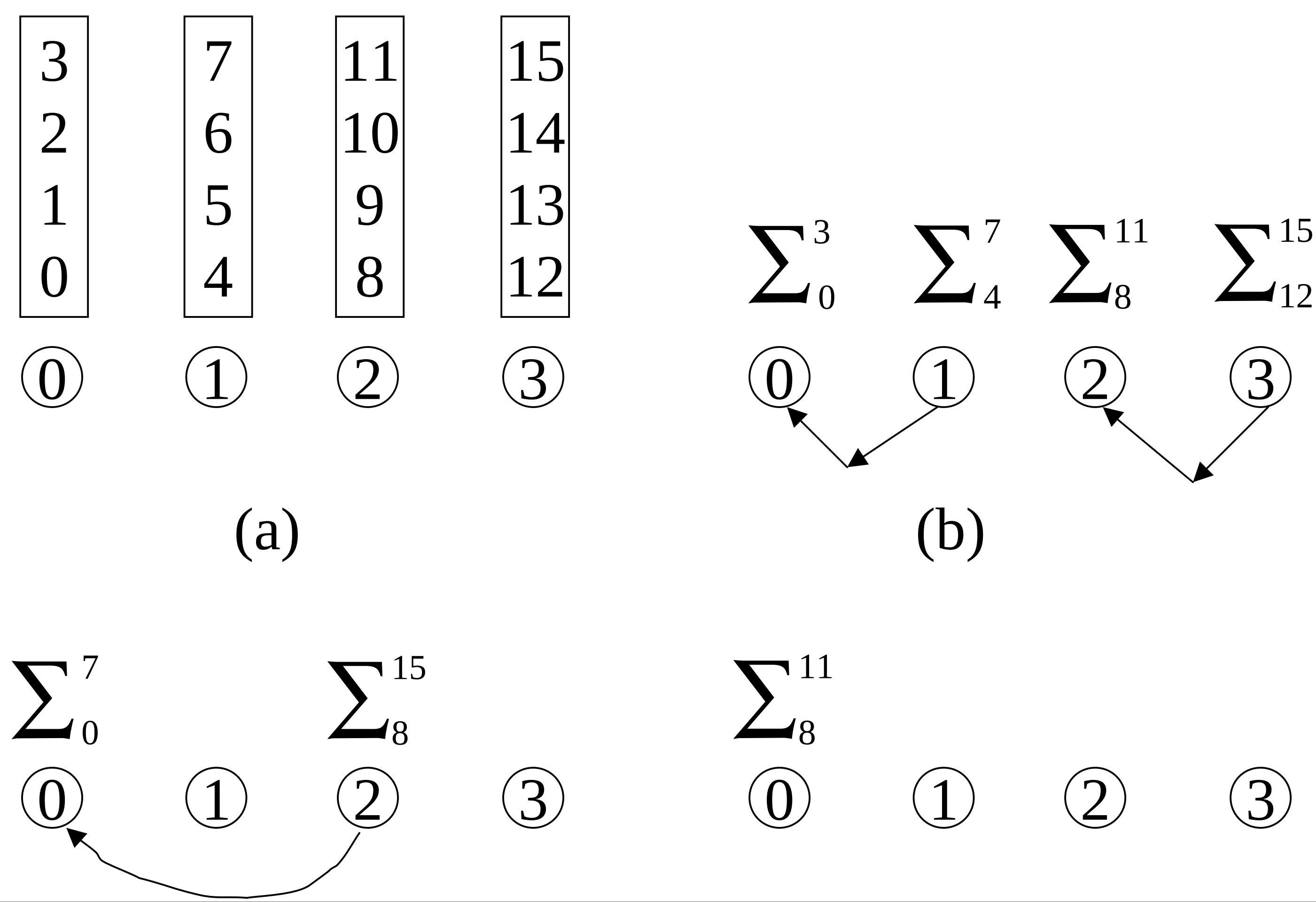
If W is constant and P increases
then E decreases

E will remain constant if $T_o(W,P)/W$ is constant
or
$W = K\ T_o(W,P)$, K = isoeffficiency constant

Hypercube Interconnection Networks for 1,2,3,4 dimensions.



Adding Numbers on a Hypercube (4 Processors)



Adding Numbers on a Hypercube

$T = \theta\ (n/p + \log P)$

Cost $= \theta\ (n + P \log P)$

$N = \Omega\ (P \log P) \Rightarrow$ cost $\theta(n)$

Let an add take 1 unit of time

Let a unit communication take 1 unit of time

Adding n/p numbers => n/p + 1

$T_p = n/p + 2 \log P$ $\qquad T_s = n - 1$

$S = n/(n/p + 2 \log P) \qquad = np/(n + 2 P \log P)$

$E = S/P = n/(n + 2 P \log P)$

θ – order of

Ω – growth rate

Scalability of Adding Numbers on a Hypercube

$N = \Omega\ (P \log P)$, E = .8 for n = 64, p = 4

E = .8 for n = 192, p = 8

=

constant efficiency

Overhead for adding n numbers in a hypercube

$T_o = pT_p - W$

$T_p = n/p + 2 \log P$

Useful work = n/p

$T_o = 2\ P \log P$

Isoefficiency of Adding Numbers on a Hypercube

$T_o = 2\ P \log P$

$W = K\ 2\ P \log P$

$K = P' \log P' / P \log P$

To maintain constant efficiency.

Isoefficiency for Finite Difference Algorithms

$E_{relative}$ (1-D grid Partition) = $t_p NxNxZ$ / ($t_c N^2 t + 2t_s P + 4NZPt_w$)

For E to be constant as a function of P

$t_p NxNxZ \sim (t_c N^2 t + 2t_s P + 4NZPt_w)$

Let us try N = P

$t_p PxPxZ = (t_c P^2 t + 2t_s P + 4PZPt_w)$

$t_p xZ = (t_c t + 2t_s/P + 4Zt_w)$

K = PxP

Isoefficiency of Finite Grid Computation

E(2-D partition) = t_cxNxNxZ/ (t_cxNxNxZ + 4xt_sxP + 8t_w NxZxSQRT(P))

t_cxNxNxZ ~ (t_cxNxNxZ + 4xt_sxP + 8t_w NxZxSQRT(P))

N = SQRT(P)

t_cxPxZ = (t_cxPxZ + 4xt_sxP + 8t_w xZxP)

K = P

Why?

Load Balancing

Definition of units of computation with identical or at least highly similar cost of execution.

Problem is usually one of data partitioning since the amount of computational cost is usually proportional to the volume of data being processed.

There is a taxonomy of load balancing problems.

- **Static, Single Resolution (Grid, Element)**
- **Static, Multi-Resolution (Grid, Element)**
- **Dynamic Single Resolution (Grid, Element)**
- **Dynamic Multi-Resolution (Grid, Element)**

Load Balancing - Continued

Methods and Algorithms

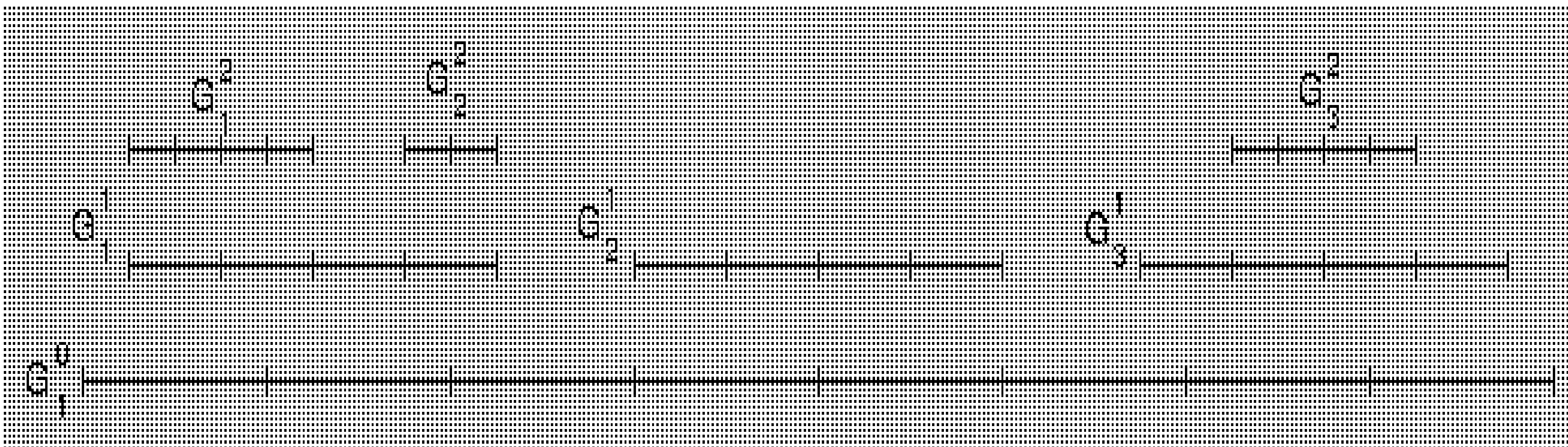
- **Grid/Mesh Based Partitioning**
- **Graph Partitioning**
  - **Metis, Parmetis, etc.**
- **Space-Filling Curves**

Example Study

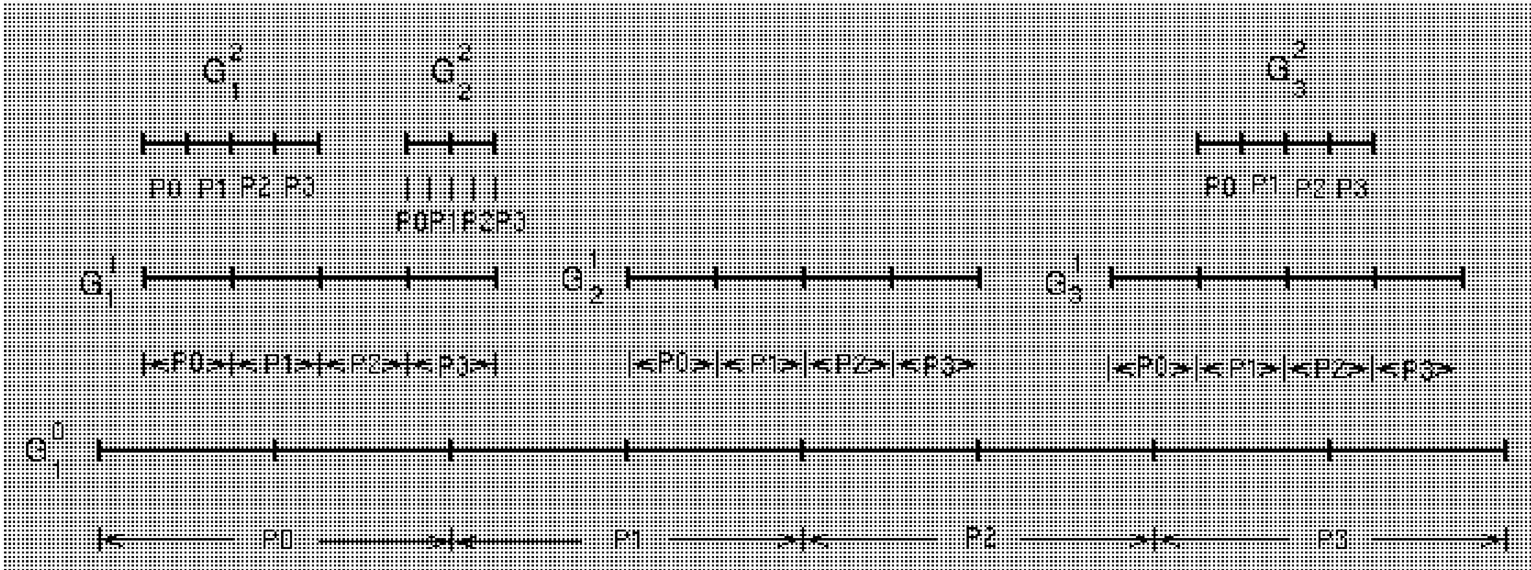
- **Dynamic, Multi-Resolution Grid**
- **Space Filling Curves**

Partitioning Issues

- Locality
- Parallelism
- Load-balance
- Cost

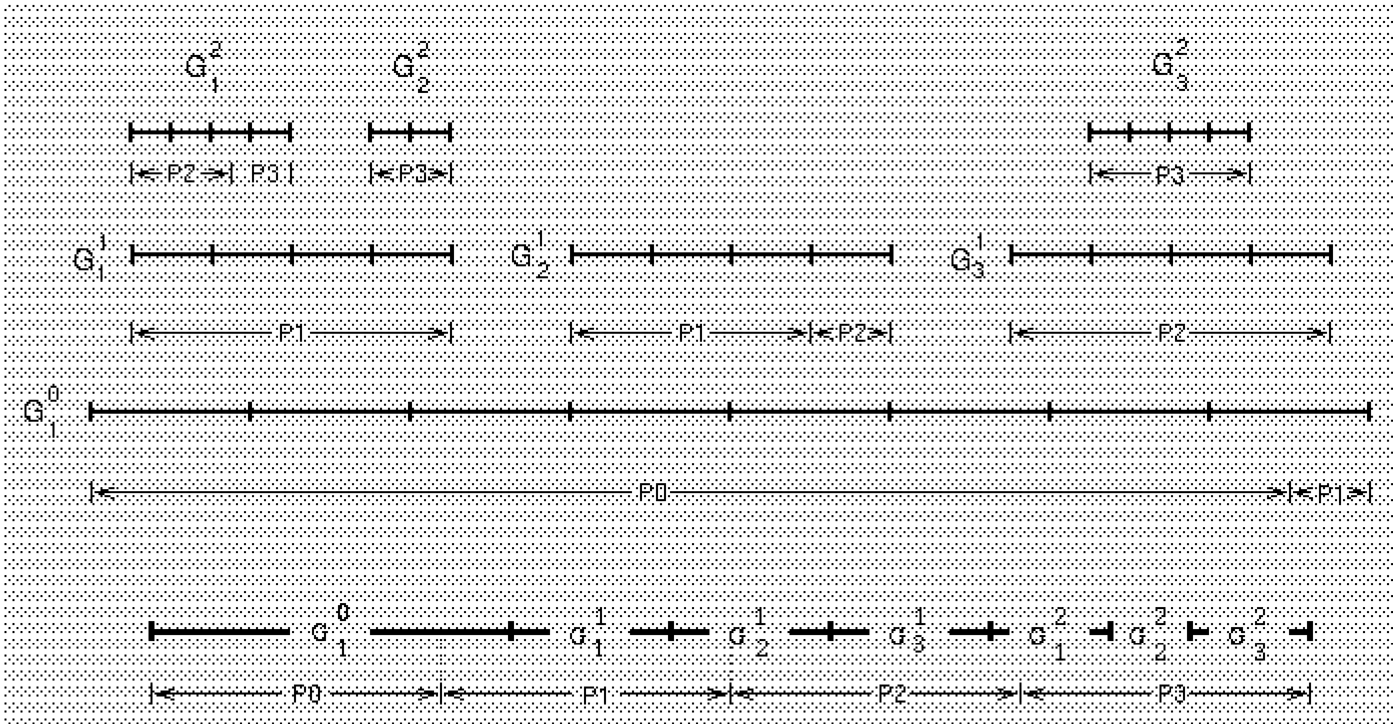


Independent Grid Distribution



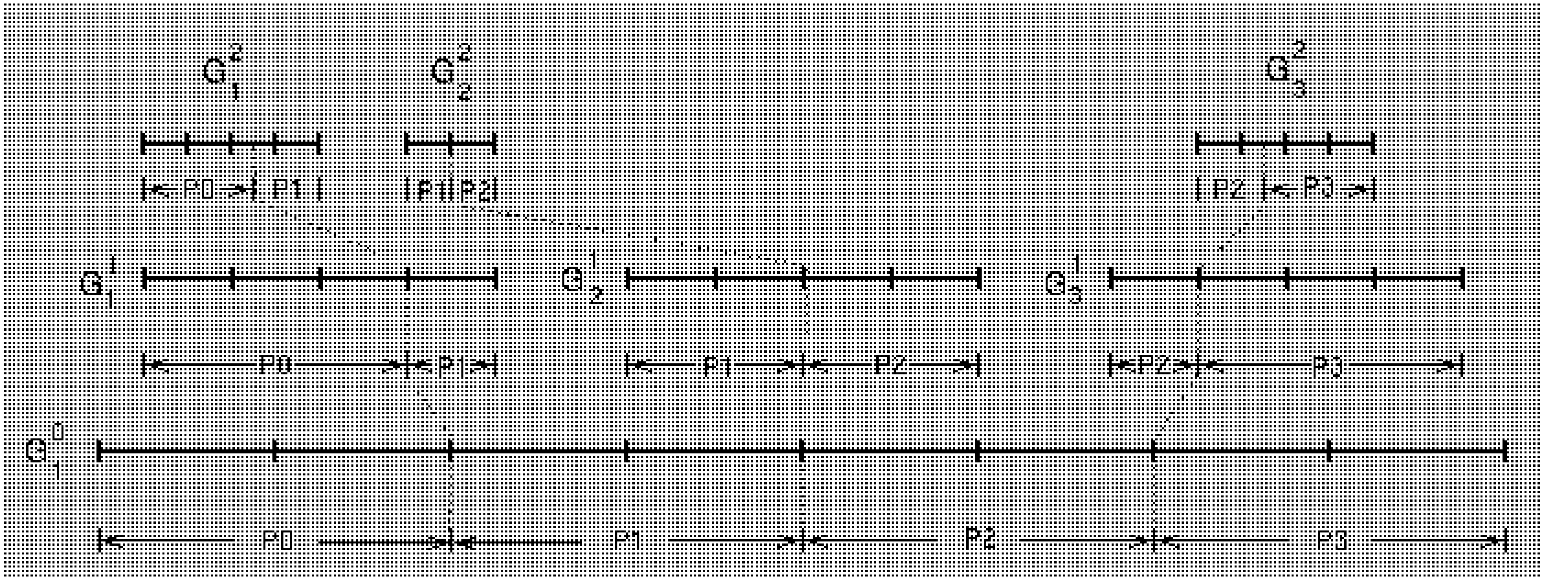
- Simple grid creation, deletion & distribution
- No work re-distribution on grid creation or deletion
- Parallelism across component grids not exploited
- Inter-grid communication bottleneck...

Combined Grid Distribution



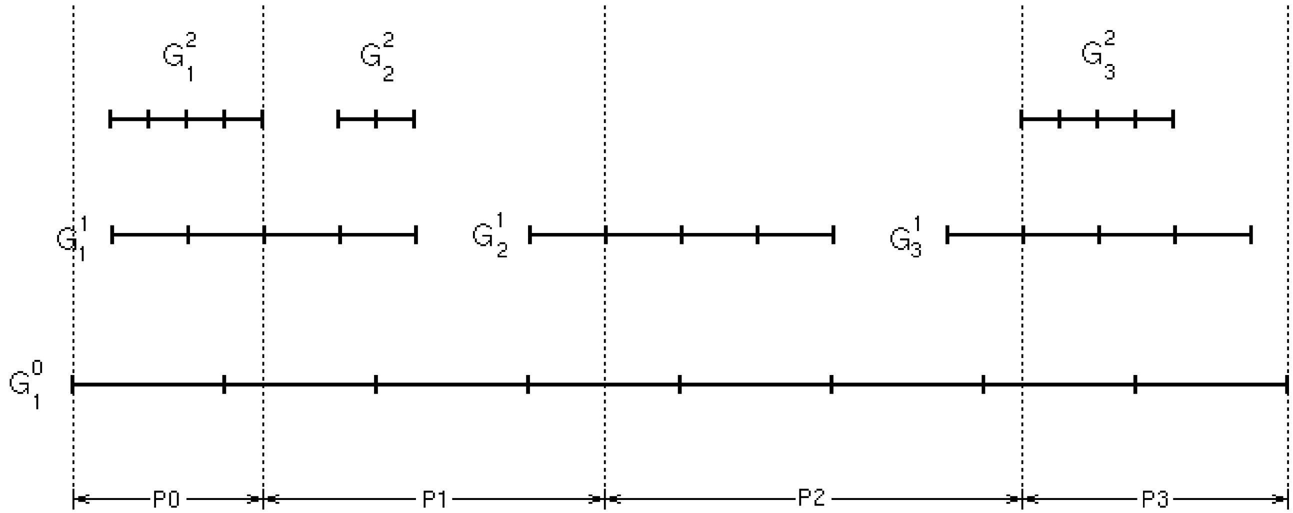
- Creation and deletion of grids can be expensive
- Parallelism within a grid level not exploited
- Inter-grid communication bottleneck reduced

Independent Level Distribution



- Creation and deletion of grids can be expensive
- Inter-grid communication bottleneck reduced
- Parallelism within a grid level exploited

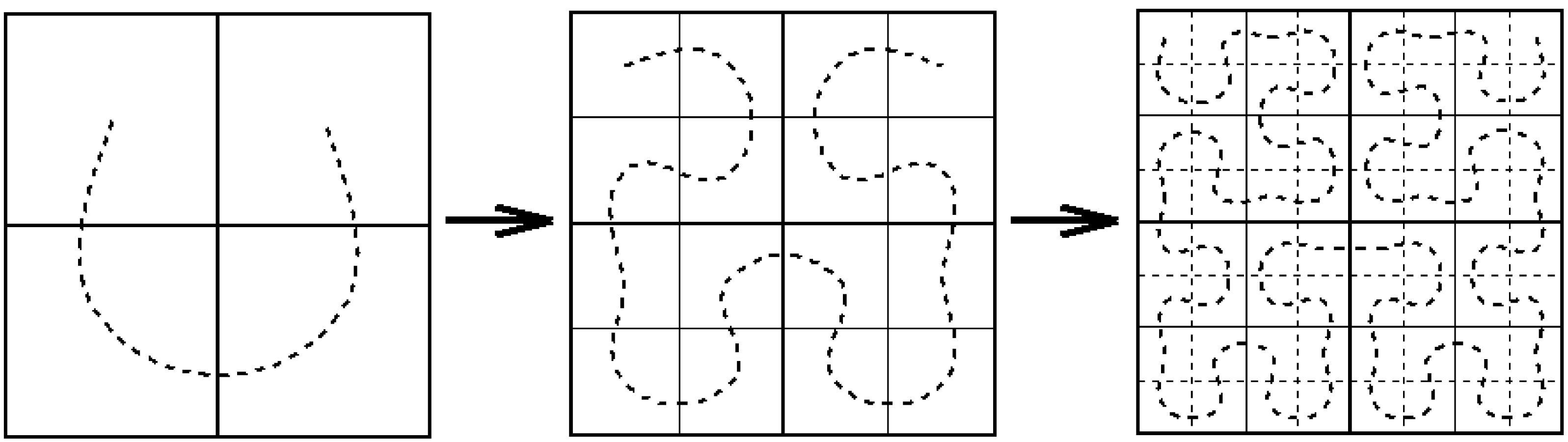
Composite Distribution



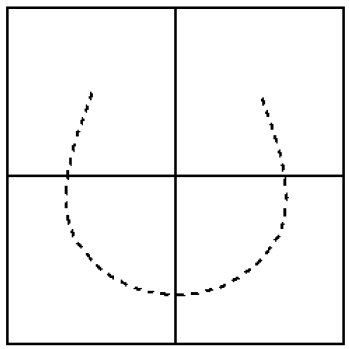
- Inter-grid communications are local
- Data and task parallelism exploited
- Efficient load redistribution and clustering
- Overhead of generating & maintaining composite structure

Space Filling Curves

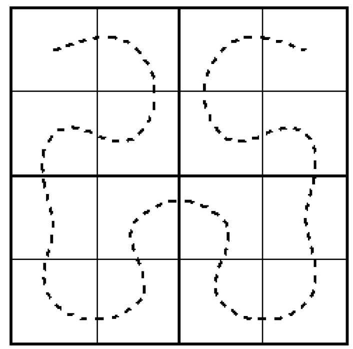
- Recursive mapping from N-d space to 1-d space
- Index locality implies geometric locality
- Locality preserved under expansion and contraction
- Computationally efficient mapping functions



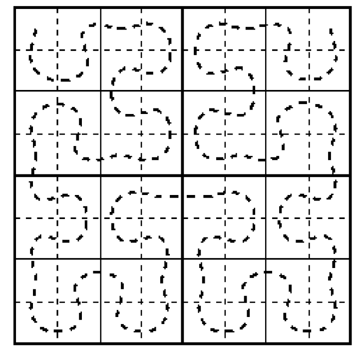
Space Filling Curves



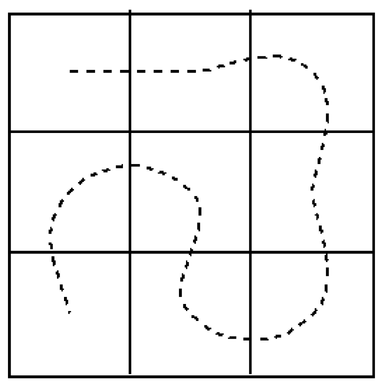
Degree 1



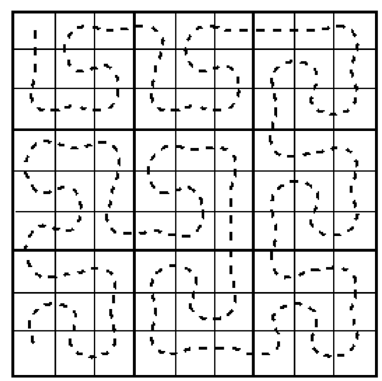
Degree 2



Degree 3



Degree 1



Degree 2

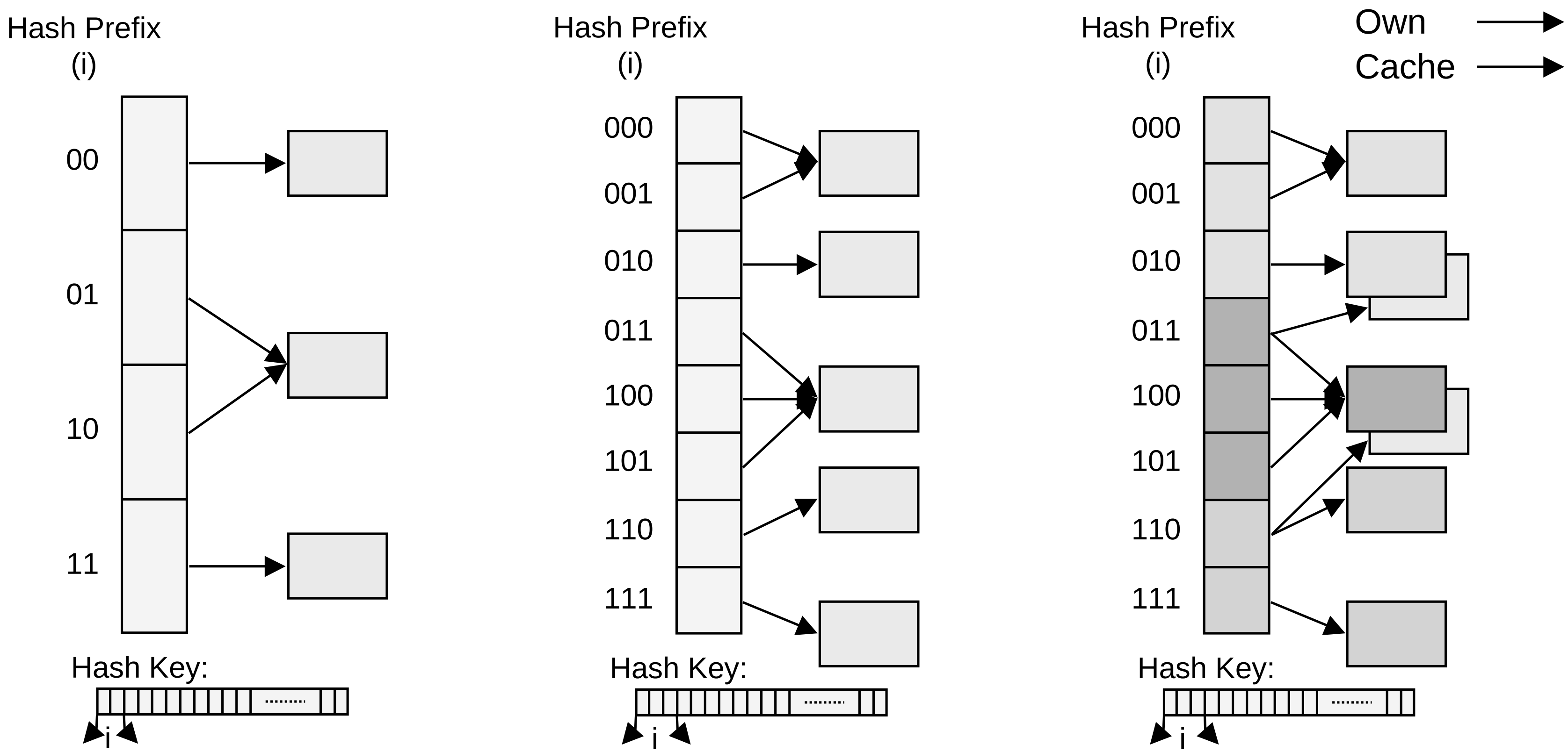
Hierarchical, Extendible Index-Space

- Encodes application locality
- Extendible, hierarchical & recursive
- Global name-space
- Linearized index space is basis for partitioning, communication, name resolution, refinement & coarsening

Storage and Access

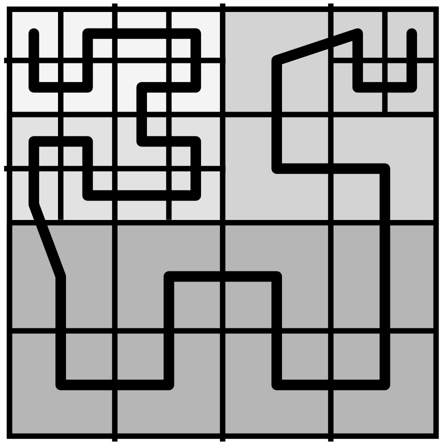
- *Extendible hashing* on the linearized global index-space
- Index locality implies storage locality
- Locality preserved through expansion & contraction
- Array access semantics

Extendible Hashing



- Dynamic hashing by splitting and coalescing buckets

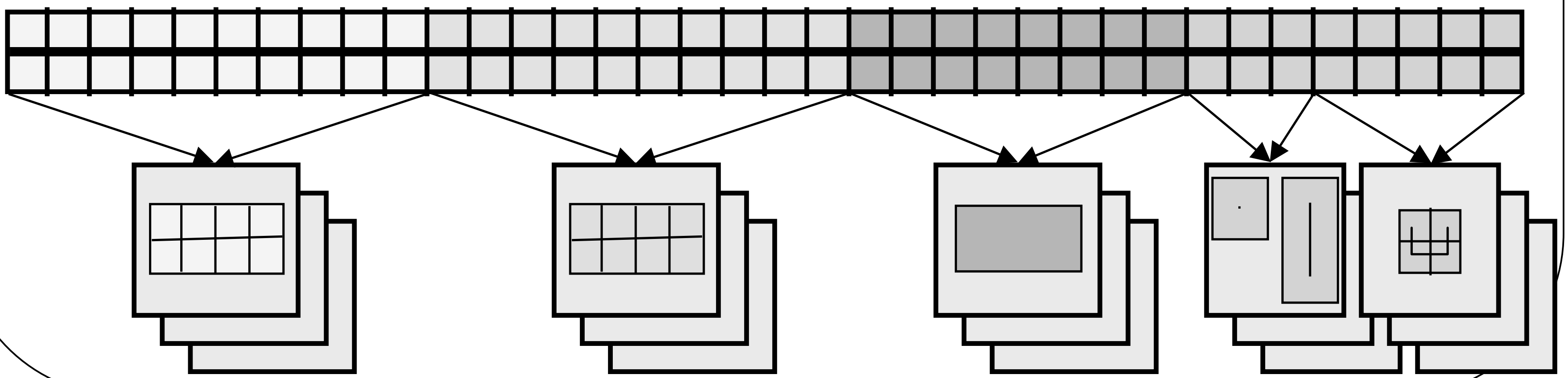
Distributed Dynamic Storage



Application Locality

→ **Index Locality**

→ **Storage Locality**



Further Topics Performance Evaluation of Parallel/Distributed Systems

- **More complex analytical models**
- **Models including contention**
- **Hierarchical Models**
- **Etc.**

Performance Monitoring and Visualization Systems

- **Pablo**
- **Paradyn**
- **Etc.**

Research in Performance Model Based Adaptive Control

- **POEMS**