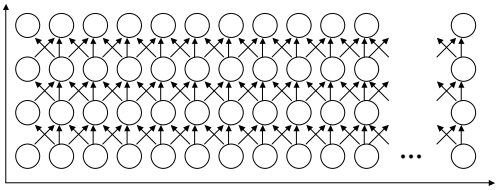
Static Clustering and Scheduling

Notions of Clustering, Scheduling, Granularity
Clustering, Scheduling for Delay Model
Clustering, Scheduling for LogP Model

Example

```
1. for (t=1;t<T;t++) {
2.   forall (k=1;i<D;k++) in parallel {
3.     δ^{t+1}[k]= f(δ^t[k -1],δ^t[k],δ^t[k +1])
4.   }
5. }
```



Mapping to distributed systems

- Two principle approaches:

1. Distribute the data
   - Each array element $\delta[k]$ is assigned to a processor
   - Computation follows
2. Cluster/Schedule the computations
   - Each computation is scheduled individually
   - Consumed array elements $\delta^t[k-1]$ $\delta^t[k]$ $\delta^t[k+1]$ and produced array elements $\delta^{t+1}[k]$ become local variables of the tasks
   - Minimize overall execution time for a cost model
     - Clustering: for arbitrary many processors
     - Scheduling: for P processors

Problem

- Given a task Graph
  - Derived from a data parallel program
  - Directly programmed
- Different concrete target machines – modeled by cost model
  - Number of processors
  - Performance of processors
  - Net performance (latency, bandwidth)
- Minimize execution time automatically for different instances

Definition Clustering

- Given a task graph $T=(N,E)$ and cost functions representing maximum costs for
  - Computation $c: N \to \mathbb{N}$
  - Communication $L: E \to \mathbb{N}$
- A clustering C is a relation $N \to (\mathbb{N}, \mathbb{N})$ where
  - C is complete in N (each node gets a starting time $t(v)$ and a processor number $p(v)$)
  - $(u,v) \in E \wedge (t, p) \in C(v) \Rightarrow$
    $\exists\, (t', p') \in C(u): t' + c(u) + L(u,v) \leq t \vee$
    $\exists\, (t', p) \in C(u): t' + c(u) \leq t$
    (starting times obey precedence relation and communication delay)
- Completion time $T(C) = \max \{t + c(v): (t, p) \in C(v) \text{ for any } v\}$

Definition Scheduling

- Given a task graph $TG=(N,E)$ and cost functions representing maximum costs c and L
- A schedule S is a relation $N \to (\mathbb{N}, P)$ where
  - S is complete in N (each node gets a starting time $t(v)$ and a processor number $p(v) \in [1 \ldots P]$
  - $(u,v) \in E \wedge (t, p) \in S(v) \Rightarrow$
    $\exists\, (t', p') \in S(u): t' + c(u) + L(u,v) \leq t \vee$
    $\exists\, (t', p) \in S(u): t' + c(u) \leq t$
    (starting times obey precedence relation and communication delay)
- Completion time:
  $T(S) = \max \{t + c(v): (t, p) \in S(v) \text{ for any } v\}$

Optimization

- An optimum clustering C_{OPT} of a task graph TG is a clustering with $T(C_{OPT}) \leq T(C)$ for all other clusterings C of TG
- An optimum schedule S_{OPT} of a task graph TG and a number of processors P is a schedule with $T(S_{OPT}) \leq T(S)$ for all other schedules S of TG and P
- It is *NP hard* to find the optimum clustering and optimum schedule, resp., for a general task graph TG (even for the simplest cost models, e.g. uniform computation costs, no communication costs)

7

Optimization *NP hard*

- In practice ok?
  - Sometimes exponential solutions are exactable if handy for practical relevant cases (minimum distribution configuration – ILP solver)
  - Unfortunately not for clustering and scheduling
- Find special cases allowing optimum solutions
- Find approximation schemata
- Find approximations with constant factor approximations
- Heuristics are ultima ratio!

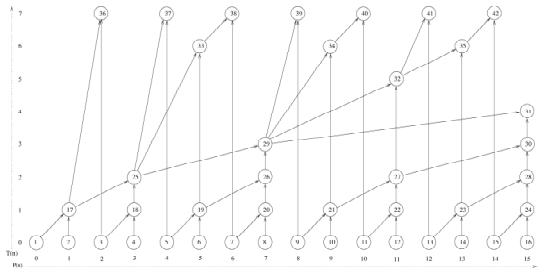
8

Example: Work opt. prefix sums

```
1.  right[0…n]=n;
2.  for (i=1;i<n;i*=2) {
3.    forall (p=0;p<n;p++) in parallel{
4.      if ((p+1) mod i = 0)
5.        right[p]=right[p-i/2]+right[p];
6.    }
7.  }
8.  for (i=n;i<0;i/=2) {
9.    forall (p=0;p<n;p++) in parallel{
10.     if ((p+1) mod i=(i/2) & p>i)
11.       right[p]=right[p-i/2]+right[p];
12.   }
13. }
```

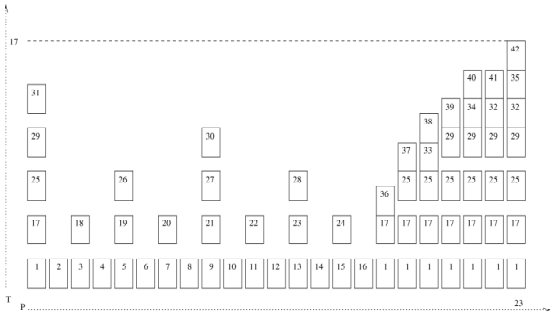
9

Example: Task graph prefix sums

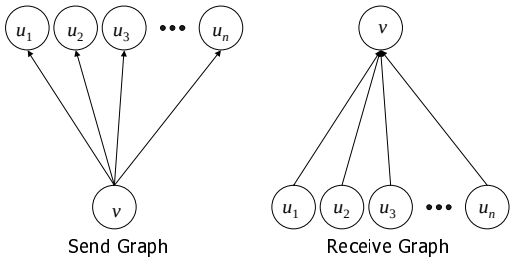


10

Clustering depicted as Gantt-Chart



Send/Receive Task Graphs

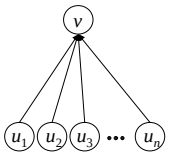


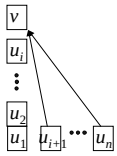
12

Cluster Receive Task Graphs

- Order all u_i non-increasingly in $c(u_i) + L(u_i, v)$
- Let $p(u_1) = p(v)$
- Stepwise add the first remaining task u_i to $p(v)$ until:

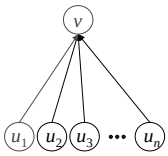
  $\Sigma_{j \in [1 \ldots i]}\, c(u_j) \geq c(u_{j+1}) + L(u_{i+1}, v)$
- Optimum solution!





Observation

- If $\min c(u_i) \geq \max L(u_i, v)$ only $p(u_1) = p(v)$
- All other tasks computed in parallel
- In general: parallel computation always pays
- Property of a task graph and cost functions under a certain cost model: granularity



Definition Granularity

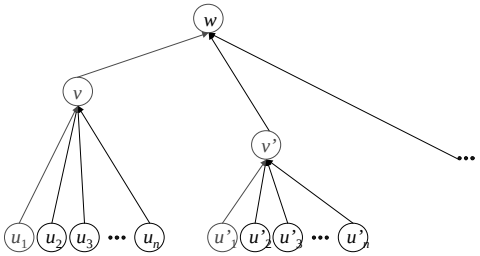
- Ratio computation / communication costs
- In order to enable quantitative conclusions, we define:
  - Granularity $G(v)$ of a task v:
    $G(v) = \min(G_{in}(v), G_{out}(v))$
    $G_{in}(v) = \min_{u_i \in Pred(v)}: c(u_i) / \max_{u_i \in Pred(v)} L(u_i, v)$
    $G_{out}(v) = \min_{u_i \in Succ(v)}: c(u_i) / \max_{u_i \in Succ(v)} L(u_i, v)$
  - Granularity $G(TG)$ of a task graph $TG = (N, E)$:
    $G(TG) = \min_{v \in N}: G(v)$
- Task graph TG is coarse grained iff $G(TG) \geq 1$
- Task graph TG is fine grained iff $G(TG) < 1$

Coarse Granularity Clustering

- For any coarse grained task graph TG a clustering $C(TG)$ with constant performance:
  $T(C) \leq 2\, T(C_{\mathrm{OPT}})$
  can be found in polynomial time
- Constructive: layer-wise
  - each task v to a separate processor
  - at starting time $t(v)\ \max_{u_i \in Pred(v)} t(u_i) + c(u_i) + L(u_i, v)$
- Proof (sketch):
  - $T(C_{\mathrm{OPT}}) \geq \Sigma_{v \in \text{ Longest Path in } TG}\ c(v)$
  - As each $c(v) \geq \max_{u_i \in Pred(v)} L(u_i, v)$ communications are smaller than $\Sigma_{v \in \text{ Longest Path in } TG}\ c(v)$

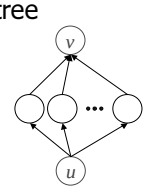
Optimum Clustering Coarse Trees

- Layer-wise, apply algorithm for receive graphs from the leaves to the root



Generalize on all task graphs

- For each output node of a TG generate tree of predecessors
- Introduces a lot of redundancy
- Find optimum clustering of each tree individually
- Unfortunately exponential if TG contains sub-structures like
- Trees have exponential many nodes



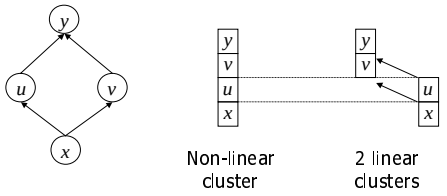
Linear Clustering

- Computing two tasks in direct precedence relation on the same processor cannot increase the overall computation time
- Any clustering constructed by merging the tasks of two processors only if all tasks are on a path in TG is called linear
- Any linear clustering C of a coarse grained TG guarantees $T(C) \leq 2\ T(C_{\mathrm{OPT}})$
- If TG is coarse grained the optimum clustering is a linear one

19

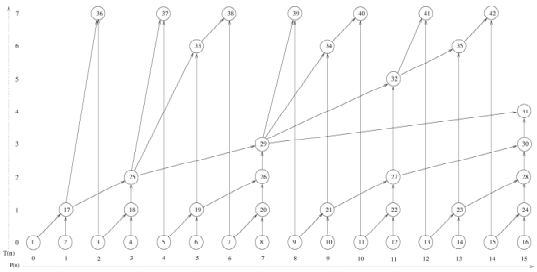
Optimum is Linear Clustering

Proof by iteration over
stepwise separation of independent tasks on a processor



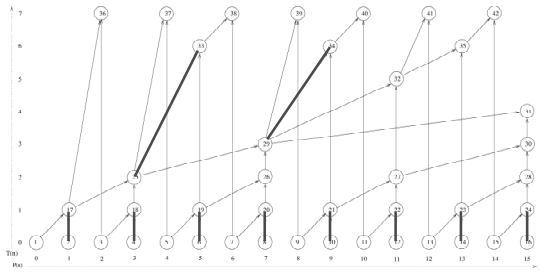
20

Linear clustering I



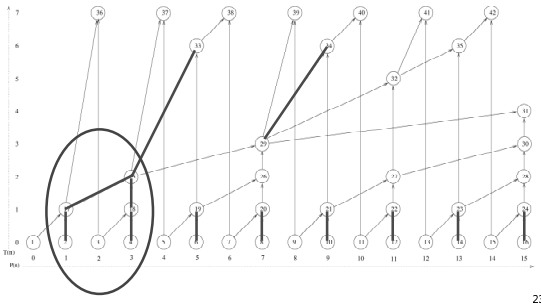
21

Linear clustering II



22

Non-Linear clustering



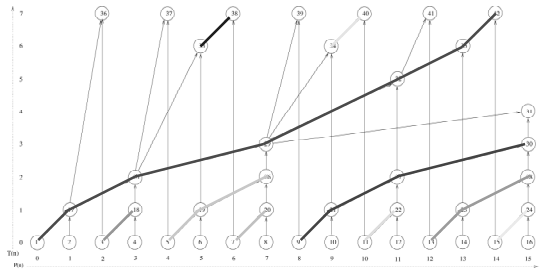
23

Heuristic: cluster longest path

- Computation dominated by longest (most expensive) path
- Idea: save communications on the longest (most expensive) path by computing its task on the same processor
- Iteratively TG remove the longest path from the until its empty
- Usually saves time and processors (both not guaranteed)

24

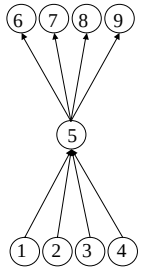
Longest Path Linear clustering



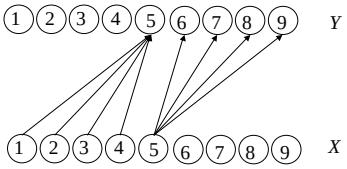
Min. Processor Linear Clustering

- Linear clustering is a node disjoined path covering Π
- Obviously $|\Pi| = P$
- Minimal path cover of $TG=(N,E)$ is maximum matching in bipartite graph:
  $M=(\{x_v \mid v \in N\} \cup \{y_v \mid v \in N\}, \{(x_u, y_v) \mid (u,v) \in E)$
- Computable in polynomial time $O(|N|^{1/2} + |E|)$

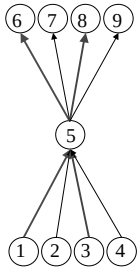
Example



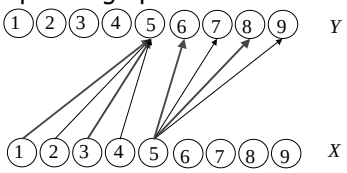
bipartite graph



Example



2 maximum matchings in bipartite graph



Linear clustering performance

- Any linear clustering C of TG guarantees $T(C) \leq (1+1/G(TG))\ T(C_{\mathrm{OPT}})$
- Proof (sketch):
  - $T(C_{\mathrm{OPT}}) \geq \Sigma_{v \in \text{ Longest Path in } TG}\ c(v)$
  - As each $1/G\ c(v) \geq \max_{u_i \in Pred(v)} L(u_i, v)$ communications are smaller than $1/G(TG)\ \Sigma_{v \in \text{ Longest Path in } TG}\ c(v)$
- Linear clustering cannot increase the overall computation time
- Bad idea if communications are expensive

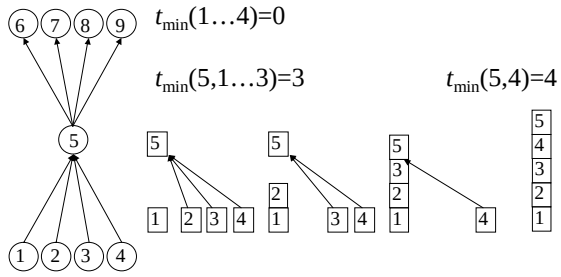
Lower Bound Optimum Clustering

- Lower bound $t_{\min}(v)$ for $t(v)$ in C_{OPT}:
  - Layer-wise in TG, begin in first layer $t_{\min}(v)=0$
  - Compute all transitive predecessors $Anc(v)$ of v
  - Assume they where direct predecessors
  - Order all $u_i \in Anc(v)$ non-increasingly in $t_{\min}(u_i) + c(u_i) + L(u_i, v)$
  - Compute $t_{\min}(v, x)$ for the x first u_i in that order:
    - Reorder $u_1 \ldots u_x$ by non-decreasing minimum starting times $t_{\min}(u_i)$
    - $c_{\min}(v,x) = \max_{i \in [1 \ldots x]} t_{\min}(u_i) + \Sigma_{k \in [i \ldots x]}\ c(u_k)$
    - $t_{\min}(v,x) = \max(c_{\min}(v,x),\ t_{\min}(u_{x+1}) + c(u_{x+1}) + L(u_{x+1}, v))$
  - Compute $t_{\min}(v)$
    - $t_{\min}(v) = \min_{x \in [1 \ldots |Anc(v)|]} t_{\min}(v,x)$

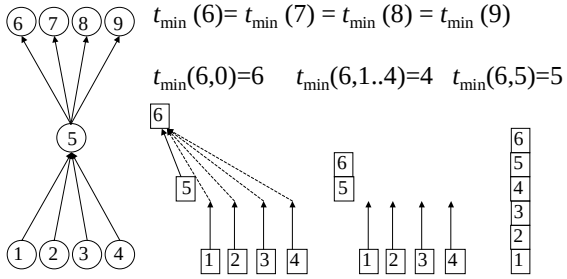
Fine Granularity Clustering

- Use lower bound $t_{\min}(v)$ for $t(v)$ in C_{OPT} for fine grained task graphs clustering
  - Compute each v on a separate processor $p(v)$
  - Let X be the smallest x making $t_{\min}(v,x)=t_{\min}(v)$
  - Set $p(u_i) = p(v)$ the first X non-increasingly in $t_{\min}(u_i)+c(u_i)+L(u_i,v)$ ordered predecessors
  - Receive results from other predecessors
  - Compute all tasks earliest
- For any task graph TG a this clustering $C(TG)$ guarantees: $T(C) \leq 2\,T(C_{\mathrm{OPT}})$

Example: L=2, c=1

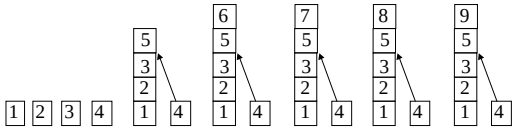


Example: L=2, c=1

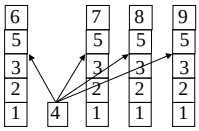


Example Schedule: L=2, c=1

Contains a lot of redundant computations



Obvious simplifications



- Only schedule output and sending tasks
- Compute each task sending its results only once.

Proof obligations

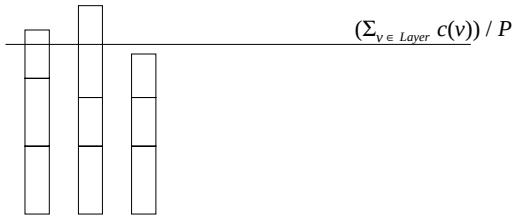
- $t_{\min}(v)$ is a lower bound for the optimum: Show that no task can be scheduled before $t_{\min}(v)$
- $T(C) \leq 2\,T(C_{\mathrm{OPT}})$ guaranteed: Show that each task can be scheduled before $2\,t_{\min}(v)$

Brent Schedule

- Reduce the number of processors by greedy packing
- Layer-wise schedule computations
  - Let $t(p)$ be the completion time of computations on processor p (initially $t(p)=0$ for all processors)
  - Schedule task v on a processor p until $t(p)$ value larger $(\Sigma_{v \in Layer}\ c(v)) / P$
- Layer-wise schedule communications (if necessary)

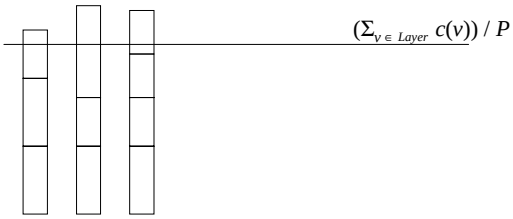
37

Schedule a layer



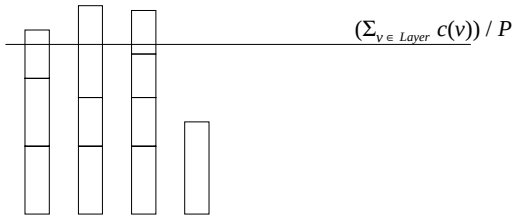
38

Schedule a layer



39

Schedule a layer



40

Performance of Brent Schedule

- Obviously requires P processors as each does work $(\Sigma_{v \in Layer}\ c(v)) / P$ or more
- Error $\leq \max_{v \in Layer}\ c(v)$
- Schedule of each layer $T(S) \leq 2\ T(S_{\mathrm{OPT}})$ since:
  - $T(S_{\mathrm{OPT}}) \geq \Sigma_{v \in \text{Longest Path in } TG}\ c(v) = \max_{v \in Layer}\ c(v)$
  - $T(S_{\mathrm{OPT}}) \geq (\Sigma_{v \in TG}\ c(v)) / P = (\Sigma_{v \in Layer}\ c(v)) / P$
- Ignoring communication costs and assuming uniform computation costs in the layers: $T(S) \leq 2\ T(S_{\mathrm{OPT}})$
- Communication cost additionally $1/G \max_{v \in Layer}\ c(v)$
- With uniform communication cost: $T(S) \leq (2+1/G(TG))\ T(S_{\mathrm{OPT}})$

41

Generalization for LogP

- Somewhat harder as communication costs processor time
- Makes it impossible to simply generalize results on delay model
- Correct, but too conservative to set:
  - Computation $c_{\mathrm{LogP}} = c+2o+(odg(v)+idg(v)-2)\max(g,o)$
  - Communication $L_{\mathrm{LogP}}(u,v) = L$

42

Send/Receive graphs (revisited)

- Optimum polynomial time solutions for delay model
- *NP hard* to find the optimum *LogP* schedule
- There is an $|N|^2$ algorithm computing a receive graph schedule with performance guarantee $T(S_{\text{LogP}}) \leq (10/3 - 1/3P))\ T(S_{\text{LogP-OPT}})$
- There is an $|N|^2 \max(\log |N|, P^2)$ algorithm computing a send graph schedule with performance guarantee $T(S_{\text{LogP}}) \leq (7/3 - 1/3P))\ T(S_{\text{LogP-OPT}})$

Receive graph schedule (sketch)

- Compute a schedule allowing exactly k receive operations for $k = 1 \ldots n$ and choose then the minimum one:
  - Order al u_i non-increasingly in $c(u_i)$
  - Schedule first k on P processors by longest processing time minimizing heuristic (greedy)
  - Schedule remaining (if any) on last processor
  - Schedule v on last processor
  - Schedule communications to last processor from first k tasks scheduled of first P-1 processors

Performance (proof sketch)

- Join of k items, must be received sequentially and compute the final task
  $2o+L+(k\text{-}1)\max(g,o) + c(v) \leq T(S_{\text{OPT}}(k))$
- Schedule tasks on P processors:
  $T(k)=(4/3 - 1/3P)\ T(S_{\text{OPT}}(k))$
  $T(S(k)) \leq T(k) + T(S_{\text{OPT}}(k))$
- Schedule last n-k tasks on processor P sequential and hence optimal
  $T(S(k)) = (4/3 - 1/3P)\ T(S_{\text{OPT}}(k)) + 2\ T(S_{\text{OPT}}(k))$
- Overall schedule selects minimum $T(S(k))$:
  $T(S) = (10/3 - 1/3P)\ T(S_{\text{OPT}}(k))$

Generalization of Granularity

- Generalize from delay model
- $L(u,v) = 2o+L+(odg(u)+idg(v)\text{-}2)\ \max(g,o)$
- Granularity $G(v)$ of a task v:
  $G(v) = \min(G_{in}(v), G_{out}(v))$
  $G_{in}(v) = \min_{u_i \in Pred(v)}: c(u_i)/\max_{u_i \in Pred(v)} L(u_i, v)$
  $G_{out}(v) = \min_{u_i \in Succ(v)}: c(u_i)/\max_{u_i \in Succ(v)} L(u_i, v)$
- Granularity $G(TG)$ of a task graph $TG = (N, E)$:
  $G(TG) = \min_{v \in N}: G(v)$
- Task graph TG is coarse grained iff $G(TG) \geq 1$
- Task graph TG is fine grained iff $G(TG) < 1$

Coarse Granularity

Computations dominate:

- Brent schedule computations
- All to all communication between two layers
- Sparse out unnecessary communication

Performance:

$4\ T_{OPT}(G) \geq T_{LogP}$

Fine Granularity

Communication dominates

- Delay communication until computation dominates
  - Allow imbalanced computation
  - Allow redundant computation
- Bundle communication
  - LogP model with communication functions
  - $L_{max}(n) < n\ L_{max}(1)$
- Within the remaining freedom
  - Balance computations
  - Eliminate unnecessary redundant computation

Results

- Balanced Trees
  $(3+\varepsilon)\ T(S_{OPT}) \geq T(S)$
- Balanced task graphs
  $(4+\varepsilon)\ T(S_{OPT}) \geq T(S)$
- Affine Index computations
  $(4+\varepsilon)\ T(S_{OPT}) \geq T(S)$

Conclusion

- Only approximations of the optimum
  - Compilers can give good results
  - Sometimes not good enough
  - Therefore programmers must be able to find better solutions for specific problems "by hand"
- Good news: "easy" new results