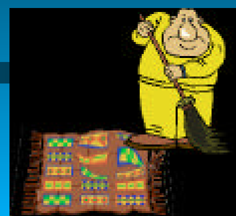
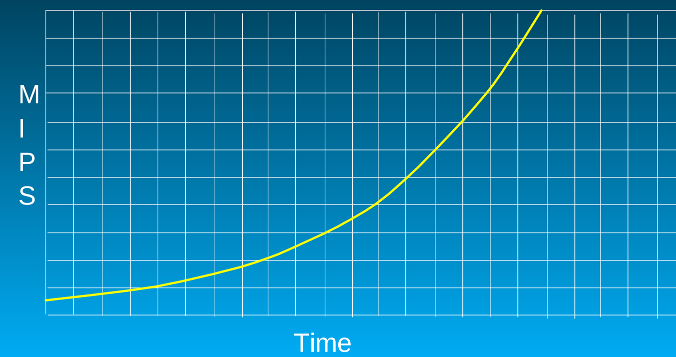
Karin Högstedt, Doug Kimelman,
VT Rajan, Tova Roth, Nan Wang,
Mark Wegman

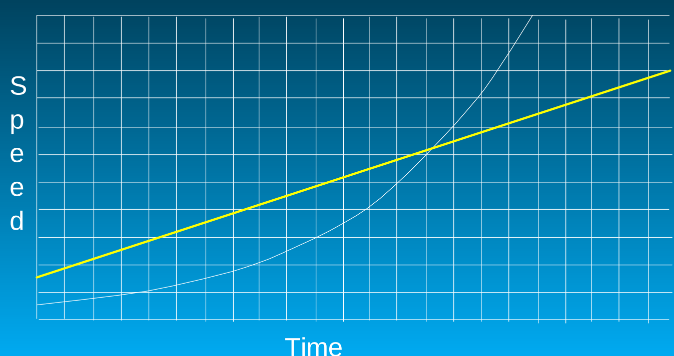
Optimizing Component Interaction



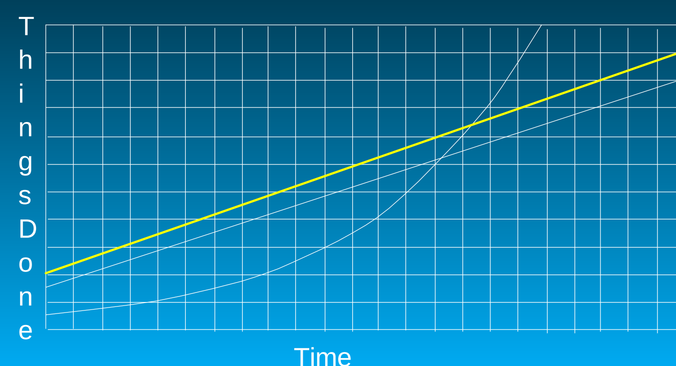
Computer Performance



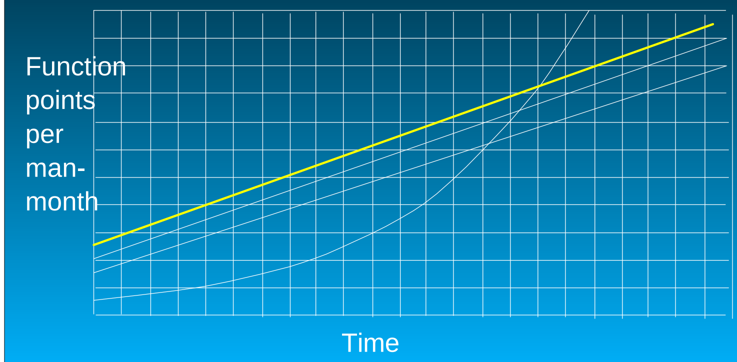
Application Performance



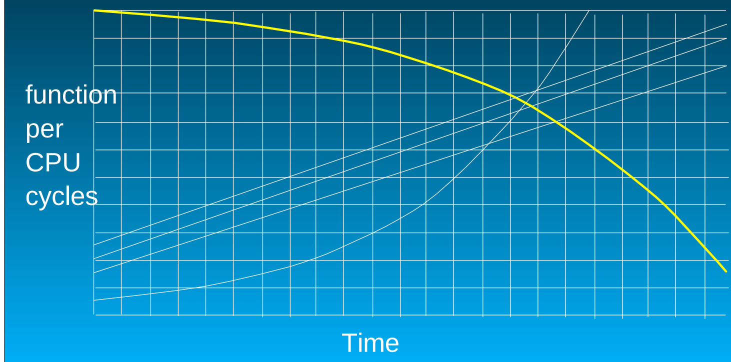
Application Function



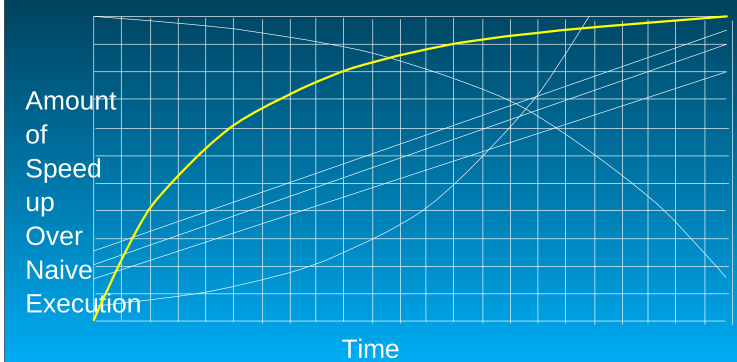
Programmer Productivity



Application Efficiency



Effects of Code Optimization



Component integration is the way people program

- **Programmers have been writing at higher and higher levels using vast libraries**
- **Separately written legacy code must be bound together**
- **Components that are designed separately will have performance problems when integrated**
  - **e. g. the library writer has no idea how his routines will be used and the user doesn't know the algorithm in the library**
- **We have studied this in the context of distribution**
  - **It is a more general problem**

Simple problem and complex ones

- One choice of component effects only itself
- Or it effects others
  - e.g. where to place a component on a network effects where another component belongs

How do you write a library?

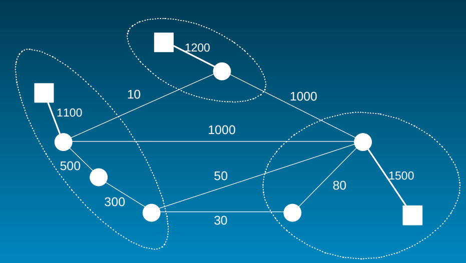
- Code multiple implementations of a class
- Write a set of instrumentation for each method
- Compose the instrumentation and implementations using a new composition rule with HyperJ
- HyperJ would make a new class for each allocation site
- Instrumentation computes values used after an initial run and a formula is evaluated to determine which implementation should be used for a given class

Note that different Objects may need different implementations
In the same program

Original Motivation

- How do you distribute entities of a distributed program to optimize its performance?
- Two entities can communicate more efficiently if they share the property of being on one particular machine
- Problem in several IBM products including VisualAge Generator, SF.
- Performance of a program written on top of SF can be affected as much as an order of magnitude by placement of objects.
- Programmers often do a poor job of placing the objects.
- Provide help to the programmers or automate the process of object placement.

The Graph Cutting Problem



VAGen Sample Program Costs

Partitioning	Cut Cost (messages Between machines)	Run Time (ms)	Run Time/Cut Cost (ms/message)
Naive	53	10.23	0.193
Manual	42	8.62	0.205
Automatic	23	4.75	0.206

How do we define a component?

	Code	Run time
Many things bundled together	Component (this def is much like a module)	
One thing	Class Definition	Entity (much like an instance)

- Components have entities bundled together which have many ways of interacting
- The code from one component produces entities that are used by the code of another
- Run time wants to bundle entities that interact most often

Notation

- Components interact through *entities*
  - via either push or pull interactions
- Entities have *properties*
  - two entities with the same properties can interact more cheaply than those with different ones
  - Which machine an entity resides on is a property
- Some entities must have certain properties
- Others can be determined based on efficiency

Example: two components that share string entities

- One component requires strings be Unicode
- The other requires Ascii

```
Ascii a,b;
Unicode c,d;
String e,f;

e=a;
f=b;
c=e+f;
d=f+e;
```

Cost of e and f being Ascii is the conversion of e+f and f+e to Unicode

Additional Motivation

- Data structures in different representation
  - Unicode Vs EBCDIC Vs ASCII
    - variables are nodes in the graph
    - Unicode, ASCII, EBCDIC are terminals
    - edges are assignment statements
  - Different Collection Class
- EJB's in different containers
- Message format in Publish-Subscribe setting?

Our Approach

- Run the program with a "typical" input.
- Trace the program using tools such as Jinsight to obtain the objects and their communications.
- Obtain the communication graph and find the optimal placement of the objects.
- Characterize the objects to allow for optimal or near optimal placement of objects during future runs.
- Help Programmer Visualize where remaining problems are.

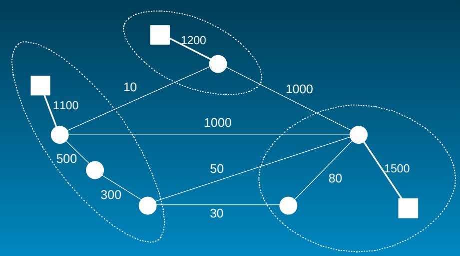
Remainder of this Talk

- A Priori optimization of a program
- A Posteriori optimization of a run of a program
- Flights of fancy over where we can go from here

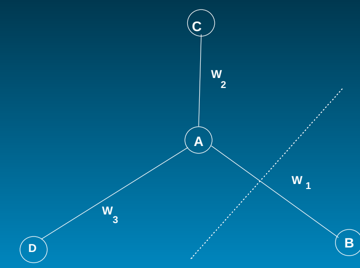
Graph Cutting is NP hard

- In our work we look for heuristics which simplify the graph, but preserve the minimum cut.
- We will ignore other constraints such as load factor - which may be important in some instances.
- We can combine our heuristics with existing algorithms.

Reprise: How to simplify this problem?

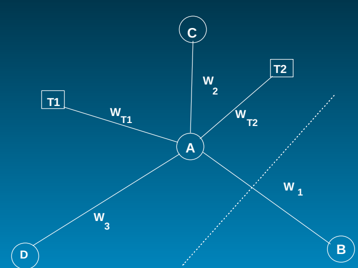


Dominant Edge heuristic



$w_1 >= w_2 + w_3$

Dominant Edge w/Terminals

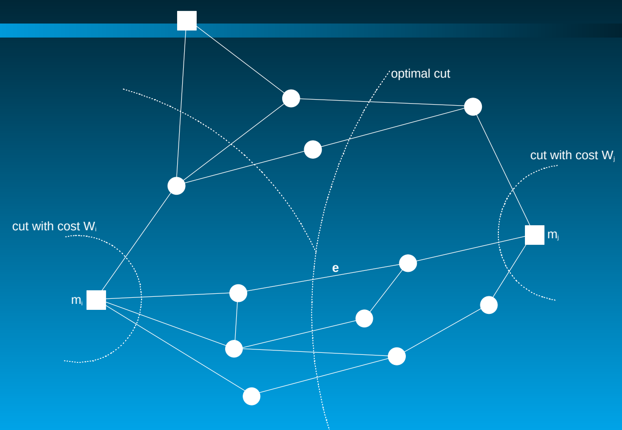


$w_1 >= w_2 + w_3 + \max(w_{T1}, w_{T2})$

Dominant Edge Application

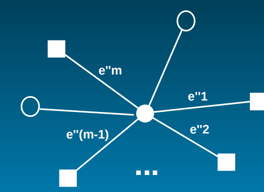
- When we discover a dominant edge we collapse the edge, and combine the nodes.
- Reduces the graph size by one - and can create new dominant edges.
- In some graphs we see over 90% reduction in graph size - by repeated application of dominant edge.
- Can be implemented to run in time O(min(degrees of the nodes)) per collapse.
- This can be done in O(E log E) time for the whole graph, E the number of edges in the graph.

Machine Cut



If $w(e) > W_2$ (second largest machine cut), it cannot be in min cut.

Zeroing

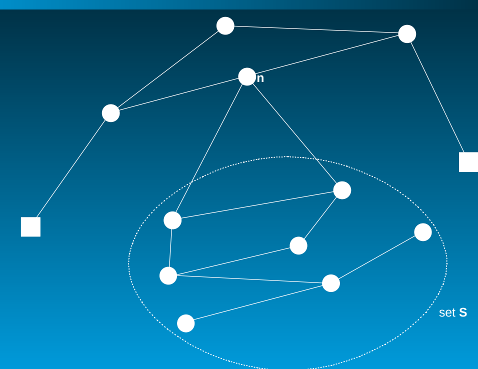


Zeroing Heuristic: The weight of edges to the Terminals 1..m can be reduced by the min(w(e"1), w(e"2), ..., w(e"m)). It helps Dominant edge and Machine Cut heuristics.

Independent Net

A graph consisting of two independent nets. One net consists of all the filled nodes, and the other net consists of all the non-filled nodes.

Articulation Point



Node n is an example of an articulation point, since all nodes in S will be separated from the rest of the graph if n is removed.

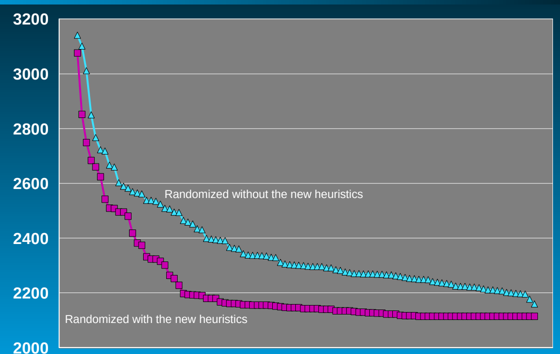
Computational Experience 1

- For several smaller graphs (20-100 nodes) from VA Gen. applications - these heuristics gave complete reductions
- One large example from pBOB (predecessor of SPECjBB2000) gave a large graph with 13,915 nodes, 32,221 edges, 404,737 messages between objects.
- The program traced with Jinsight.
- Dominant edge (w/terminals) heuristic reduced the graph to 1695 nodes and 7494 edges.
- Zeroing and machine cut heuristic reduced the graph to 1597 nodes and 3990 edges.
- Dominant edge heuristic reduced the graph to 39 nodes and 110 edges.
- Articulation point heuristic reduced the graph to 6 nodes and 5 edges (5 terminal nodes).
- Dominant edge reduced the graph to 5 terminal nodes.

Computational Experience 2

- Another run of pBOB, focusing on the transaction part of it.
- Graph with 3543 nodes and 5485 edges.
- Dominant edge heuristic reduced it to 198 nodes and 774 edges.
- Articulation Point heuristic reduced it to 161 nodes and 660 edges.
- Then had to use randomized reduction or branch and bound technique.
- Typically 6-20 collapses using random - and then these heuristics reduced the graph completely.
- Randomized reduction gave a probable minimum.
- Distribution of nodes was more uniform - 672, 1055, 689 and, 1127 nodes on each of the four machines.
- The randomized algorithm converged significantly more rapidly when we combined it with our heuristics.

Results of Randomized: With and without new heuristics



Comparison of our Partitioning Algorithm

data	Spec1	Spec2	Spec3	Spec4
Number of entities	1,972	3,317	6,197	11,478
Number of edges	2,844	4,896	9,444	17,878
Number of messages	29,323	53,954	109,503	210,889
Weight of optimal cut	1,418	2,611	5,288	10,901
Weight w/o Dalhouse	1,418	2,642	5,437	10,914
Weight Schloegel's algorithm gets	2,061	3,710	5,754	13,070

Related Work

- Distributed Application partitioning problem is related to Graph cutting - H. Stone 1977.
- There has been work using various heuristics to obtain approximate solution, e.g. Stoyenko et. al.
- When there is only two terminals we can solve the problem using max-flow (Ford-Folkerson).
- When there are more than two terminals, the problem is NP-hard - Dahlhaus et. al.1994.

Conclusion about A Priori optimization

- Even though the multi-terminal graph cutting problem is NP-hard, these heuristics can significantly reduce the graph.
- In many cases they yield optimal results.
- Even when they do not completely reduced the graph, they enhance the performance of other algorithms.
- We would like to explore the applicability of these heuristics to other graph cutting problems, such as the ones from network problems.

Using Dynamic Information to Distribute OO Programs

- Components are assembled but their developers often know nothing about what the components will be connected to
- We have experimented with automatic distribution involving:
  - Running the program determining how often one object communicates with another
  - Partitioning the resulting graph
  - Characterizing the objects which end up on the different machines

Characterization: Basic Idea

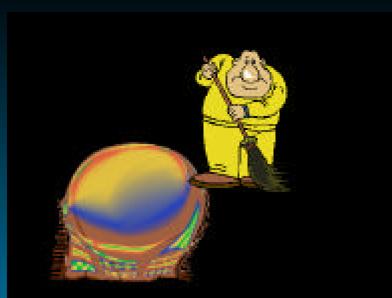
- For each class of objects or each allocation site, construct a strategy for determining properties for entities at create time
- Possible strategies
  - All objects of that a given class have the same property
  - Use machine that the creation was done on to determine where it should be allocated (has same property as the creator)

Characterization Greedy Algorithm

- Partition the entities optimally
- For each class determine cost of moving all instances of the class to a terminal
- For each class determine cost of putting the instances of a class on the same terminal as their creator
- Unify elements of the most obvious class with either terminal or creating entity

Experience with Characterization

- Class objects and factories need to be replicated
- Benchmarks don't contain all the information needed
  - Creator information is not present during the part of the run that is the benchmark
- If we have four warehouses and four customers, class is not enough
- Except when information lost during benchmarks we have succeeded in the few cases we have attempted
  - Greedy has worked optimally



Flights of Fancy Section

Other important techniques

- Replication -- If an entity is not going to be modified, just make a copy with the alternate property
- Caching -- convert it from one property to the other only on demand and keep it with that property until needed with the other
  - Data structure caching instead of data motion caching
  - This is one way of discussing data movement
  - David Bacon has looked at this for strings
- Characterization needed to determine if the overhead is worth it.