EECS 527 -Project 2

Topic

Implementing Flat and Multi-Level Fiduccia-Mattheyses Algorithms For Hypergraph Partitioning

It is recommended that you finish Homework 2 before starting this assignment.

The size of tar.gz should be <100kb, do not include binaries or large benchmarks.
All your source codes should be there, with a Makefile and a README file (see below).

Technical requirements

Your programs must be written in C or C++ and compile with the g++ compiler on Linux and Solaris. You are encouraged to use C++ and STL, but this is not required for mini-project 2. Your implementations of FM and MLFM must be entirely written by you (you cannot copy anyone else's code), but you may read relevant implementations in UCLA Physical Design tools and adapt ideas from there to your implementations. Your implementation of FM must take linear time per pass.

The following versions of g++ are accepted:
g++ 2.95.2 - 2.95.4, g++ 3.0, g++ 3.1 and g++ 3.2
The version of g++ that you used should be mentioned in the README file and the Makefile.

Your programs must be able to produce "reasonable" balanced 2-way partitions for all 18 ibm benchmarks on Linux and Solaris workstations with no more than 512Mb RAM. "Reasonable" is defined as "not too far from the second best submission by students in this course".

Algorithms and Optimization Objectives

Your programs must minimize net cut.
You must use the same IBM benchmarks as in Homeworks 1 ad 2, including vertex weights. The min-cut optimization is subject to the 42%-58% balance constraint, as in Homeworks 1 and 2. The resulting partitions should be saved to the file output.txt, on every line there should be a 0 or 1, according to where a given vertex is assigned.

Your programs will be judged based on whether they run well, based on their runtime and solution quality.

In terms of algorithms, you must implement the Fiduccia-Mattheyses algorithm and its multi-level extensions (MLFM). Those must be evaluated empirically in terms of Pareto curves and compared

to respective implementations from UCLA Physical Design Tools, dart-throwing and your simple partitioning algorithm from homework 1.

You must compare several clustering schemes, including
- entirely random (pair up vertices at random),
- random neighbors (pair up only adjacent vertices chosen at random),
- most connected neighbors (as discussed in class and in papers by G. Karypis et al. -- there can be many variants)

Is it true that using *any* of these schemes in the multi-level framework allows one to find better partitioning solutions than using flat FM and spend less time doing so?

Documentation

The README.txt file (or README.ps or README.pdf) enclosed with your implementation must contain your name and student ID. It must explain explain how to run your program, mention which data formats are expected at the input and how the output is saved. It must explain your design decisionsand non-trivial implementation details that are key to the (good) performance of your implementations.

Give runtime for the first pass of flat FM on each of the benchmarks and plot those runtimes against the input size (in bytes) to show that your FM takes linear time per pass.

Give the average number of passes used by flat FM on each of 18 IBM benchmarks.
Give the average number of FM passes per level (excluding top-level partitioning) for the same benchmarks.

Give runtime breakdown for your program in percent for major parts, such as reading input, clustering, top-level partitioning and refining. Give a similar breakdown of memory used. (Note that you can use these profiling data to optimize your program).

Give empirical cuts and runtimes on the 18 IBM benchmarks for your best implementations of FM and MLFM and compare them to algorithms from Homework 2 in terms of Pareto curves.

Rank the 18 benchmarks in the order of difficulty by seeing how often best solutions are reached if you run 100 independent starts of your FM and MLFM. Are these rankings the same for FM and MLFM?

==========