CS258f Project Report
Kenton Sze
Kevin Chen
06.10.02
Prof Cong

Multi-level Quadratic Placement for Standard Cell Designs

Project Description/Objectives:

The goal of this project was to provide an algorithm to produce an optimized VLSI placement of a circuit using a multi-level quadratic placement scheme of our own choice. We were to select and implement a C++ coarsening/uncoarsening scheme as well as a quadratic initial placement and refinement algorithm. A program (Domino) was provided to generate a detailed placement from our global placement. Our objective was to minimize the total wirelength using the half-perimeter bounding box model as well as the total runtime of our placement scheme. For comparison, we were given seven benchmark circuits on which to run our algorithm as well as the results of Gordian and Domino.

General Overview of Existing Placement Methods:

With regards to the problem of circuit placement, we find that since it is a classical problem in VLSI physical design, various effective placement tools have been proposed [5,6,7,8]. Simulated Annealing is one well-known placement method which can handle complex design constraints. However, the annealing process [6] is slow and moves only a small numbers of cells in each step. Also, as the design size increases, its runtime and quality do not follow as well.

Another traditional approach is quadratic programming-based placement techniques [5] which are very efficient. These provide a good initial placement for many situations. For a cell count greater than 100,000, the multilevel approach for circuit placement [8] demonstrates great efficiency. The idea is to cluster the original graph using a coarsening scheme (mentioned below) and thus reducing the problem size (number of distinct elements in the graph). We can then solve the problem efficiently at the coarsest level and perform declustering and refinement at each level until the graph is uncoarsened to its initial state.

When selecting an existing coarsening scheme, we had many choices which included Edge Coarsening (EC), Hyperedge Coarsening (HEC), Modified Hyperedge Coarsening (MHEC), and FirstChoice (FC). From the above schemes, detailed in "Multilevel k-way hypergraph partitioning", we selected FirstChoice as it was the scheme derived from the others and was created to solve certain problems plaguing the other schemes, such as the fact that both EC and HEC schemes can cause the destruction of natural clusters in the hypergraph.

Description of methods:

Through the use of a C++ STL program, we were able to model and manipulate the circuit data (nets, cells, groups, etc.). We began by using the given parser program to read in the circuit files. We then created our data structure and member functions in the parser source files and then created separate files for the various procedures such as coarsening, refinement, and final placement.

For the coarsening/uncoarsening scheme, we used the FirstChoice algorithm detailed in the papers by Karypis et. al [3,4]. In this scheme, we traversed each cell in the circuit and grouped it with a cell within the same net which, when combined, produces the maximum edge-weight. When a pair of cells are grouped, they become a single element. Thus, the node count in the circuit decreases due to coarsening. When we reach a pre-specified level of coarsening (limited by the number of distinct elements after grouping), we can begin the initial placement. After the initial placement, we can uncoarsen (expand) the groups in the reverse order in a series of steps and then use refinement on the elements. When the uncoarsening/refinement is complete, this results in our global placement result, which we then run through Domino to generate our final detailed placement result.

For our placement scheme, we used quadratic minimization of the wirelengths within each cell using a center of mass calculation. For each net, we calculated a center of mass and attempted to minimize the wirelengths connecting the center of mass to the individual cells. We used a quadratic solver with the conjugate gradient method with preconditioner to solve the problem efficiently. After examining the open-source LASPACK software, we decided to implement our own quadratic solver as it provided us with more flexibility and allowed us to adapt it well to our problem. We tried both the Jacobi, SSOR, and ILU preconditioners. We found experimentally that the ILU converged much faster for the number of iterations in the conjugate gradient method. However, the Jacobi preconditioner was much faster than the ILU and SSOR in terms of CPU runtime and consumed less memory.

After solving the quadratic placement, we then used a greedy approach for slot assignment, where we first sorted the x-coordinate values of the cells to place them into their respective columns. Then, we sorted the cells in each column by their y-coordinate values. Finally, we utilized a discrete local refinement to further reduce half-perimeter wirelength by examining the neighbors to a cell and swapping locations with it if it causes the maximum reduction in wirelength among all neighbors. For our refinement algorithm, a neighbor is defined as a cell within the square area around a particular cell of arbitrary distance. We do this repeatedly at the center position until there are no possible reductions in wirelength due to a swap. This creates a tendency for all cells to move to their optimum positions within this local area.

Circuit characteristics:

Circuit	#cells	#nets	#pads	Max cell degree	Max net size	Cell ht	# rows
avqsmall	21854	30038	64	8	4042	116	80
avqlarge	25114	33298	64	8	4042	116	86
ibm07	45639	48117	287	98	25	10	151
ibm10	68685	75196	744	137	41	10	185
ibm14	147088	152772	517	270	33	10	1008
ibm17	184752	189581	743	81	36	10	303
ibm18	210341	201920	272	97	66	10	324

Gordian/Domino results:

Circuit	#cells	#nets	Gordian w ire	Initial Domino	After Domino	Gordian runt	Domino runti	Total runtime	Total runtime
avqsmall	21854	30038	1.30E+07	1.22E+07	1.13E+07	1312.417	399.267	28.528	0.48
avqlarge	25114	33298	1.45E+07	1.35E+07	1.26E+07	1631.467	307.1	32.309	0.54
ibm07	45639	48117	1.33E+07	1.18E+07	1.09E+07	10843.45	180.967	183.740	3.06
ibm10	68685	75196	2.26E+07	2.00E+07	1.88E+07	23868.2	320.1	403.138	6.72
ibm14	147088	152772	5.03E+07	4.31E+07	4.08E+07	60208.35	1181.533	1023.165	17.05
ibm17	184752	189581	8.93E+07	7.36E+07	6.79E+07	139398.4	1714.6	2351.883	39.20
ibm18	210341	201920	8.88E+07	5.78E+07	5.37E+07	202217.3	2056.867	3404.570	56.74

Our results:

Circuit	Initial Domino	After Domino	program runtime:	Domino runtime	Total runtime (min)	Total runtime (hrs)	
avqsmall	1.52E+08	5.35E+07	2240	516	45.933	0.77	*
avqlarge	1.69E+08	5.92E+07	2520	324	47.400	0.79	*
ibm07	2.18E+07	1.90E+07	4506	136	77.367	1.29	+
ibm10	4.30E+07	3.74E+07	11059	252	188.517	3.14	+
ibm14	1.01E+08	8.72E+07	15328	800	268.800	4.48	+
ibm17	1.63E+08	1.44E+08	40272	1320	693.200	11.55	+
ibm18	1.47E+08	1.30E+08	48711	1569	838.000	13.97	+

* 20 iterations of generation of new placements in Domino

+ 10 iterations

Comparison with Gordian/Domino pair:

Circuit	Gordian wirelength	Our init Domino	Percentage:	After Domino	Our final:	Percentage:	G/D runtime:	Our runtime:
avqsmall	1.30E+07	1.52E+08	1.17E+01	1.13E+07	5.35E+07	4.72E+00	0.48	0.77
avqlarge	1.45E+07	1.69E+08	1.17E+01	1.26E+07	5.92E+07	4.70E+00	0.54	0.79
ibm07	1.33E+07	2.18E+07	1.64E+00	1.09E+07	1.90E+07	1.75E+00	3.06	1.29
ibm10	2.26E+07	4.30E+07	1.90E+00	1.88E+07	3.74E+07	1.99E+00	6.72	3.14
ibm14	5.03E+07	1.01E+08	2.01E+00	4.08E+07	8.72E+07	2.13E+00	17.05	4.48
ibm17	8.93E+07	1.63E+08	1.82E+00	6.79E+07	1.44E+08	2.12E+00	39.20	11.55
ibm18	8.88E+07	1.47E+08	1.66E+00	5.37E+07	1.30E+08	2.42E+00	56.74	13.97

Discussion of results:

From these results, we see that our obtained wirelengths were close to two times the Domino/Gordian pair for the ibm circuits. The avq circuits produced about four times the wirelength and required a longer runtime. This is due to the fact that the avq circuits have a much larger net size which causes the quadratic approximation to become inadequate. This produces a denser matrix, which is much more calculation-intensive with regards to computation and results in the slower runtime as well as less-than-optimal wirelength. However, in contrast, our runtimes for the ibm circuits were well below that of the Gordian/Domino pair.

The FirstChoice coarsening scheme was implemented and integrated in our program, however it was not used in our runs due to extremely long runtimes. Given more time, we would have reduced the runtime required in the coarsening stage and this would have helped to improve our results (both wirelength and runtime) in the avqsmall and avqlarge benchmark circuits.

System details:

Gordian/Domino results for all circuits:

Machine name: whale.cs.ucla.edu

- SunOS whale 5.7 Generic_106541-18 sun4u sparc SUNW,Ultra-4
- Memory: 4096M real
- Processor 0 - The sparcv9 processor operates at 400 MHz
- Processor 1 - The sparcv9 processor operates at 400 MHz
- Processor 2 - The sparcv9 processor operates at 400 MHz
- Processor 3 - The sparcv9 processor operates at 400 MHz

Our algorithm for all circuits:

Machine name: symphony.cs.ucla.edu

- SunOS symphony 5.8 Generic_108528-10 sun4u sparc SUNW,Sun-Blade-1000
- Memory: 2560M real
- Processor 0 - The sparcv9 processor operates at 750 MHz
- Processor 1 - The sparcv9 processor operates at 750 MHz

Conclusion:

Although our wirelengths were close to two times that of the Gordian/Domino pair, there was still a benefit in our greatly decreased runtime. However, we felt there was room for improvement in wirelength minimization. Specifying a larger local refinement radius would also decrease wirelength at the expense of longer runtime. Another possibility is modifying the FirstChoice coarsening algorithm for improved efficiency. This would yield a faster runtime and reduced wirelength.

Our current algorithm produces extremely fast runtimes at the expense of increased wirelength. This can result in a trade-off between reduced wirelength and reduced program runtime. In the future, we can implement a scheme to further reduce the wirelength, but causing an increase in runtime. We can then allow the user to make the decision as to whether runtime or wirelength minimization is more important.

Our project source code can be found at the following location:

/u/guest/knchen/CS258/finalsource

Another copy is also located at the following locations in the event that our guest CS accounts are deactivated before the three week period following the end of the quarter:

/w/grad.01/ee/kevinc/www/cs258f

or

http://www.seas.ucla.edu/~kevinc/cs258f/

and at

/v/usera/nksze/project/finalb4pastecode

A description of our source files is as follows:

parser.h (298 lines)	The header file for our source code
parser.cpp (294 lines)	Contains the member functions for our data structure
MLmain.cpp (250 lines)	The location of our main() function
coarsening.cpp (604 lines)	Contains the FirstChoice coarsening scheme
refinement.cpp (623 lines)	Contains the refinement algorithm
fplace.cpp (142 lines)	Contains the final placement algorithm
my_math.cpp (531 lines)	Contains the custom math functions

Total program length: 2742 lines

Acknowledgements:

Many thanks to Xin and Professor Cong for their support and advice regarding the project.

Bibliography

[1] G. Sigl, K. Doll, and H. Spruth, GORDIAN user's guide, 1995.

[2] K. Doll and B. Riess, DOMINO user's guide, 1995.

[3] G. Karypis and V. Kumar, Multilevel k-way hypergraph partitioning, *Proc. Design Automation Conference*, pp.343-348, 1998.

[4] G. Karypis, V. Kumar, and R. Aggarwal, Multilevel hypergraph partitioning: Applications in VLSI Domain, *IEEE Transactions on VLSI Systems, Vol. 7, No. 1, March*, pp.69-78, 1999.

[5] J. M. Kleinhans, G. Sigl, F.M. Johannes, and K.J. Antreich, GORDIAN: VLSI placement by quadratic programming and slicing optimization, *IEEE Trans. on Computer-Aided Design*, pp.356-365, 1991.

[6] M. Sarrafzadeh and M. Wang. "NRG: Global and Detailed Placement". In *International Conference on Computer-Aided Design*. IEEE, November 1997.

[7] W. J. Sun and C. Sechen. "A Loosely Coupled Parallel Algorithm for Standard Cell Placement". ". In *International Conference on Computer-Aided Design*, pages 137-144. IEEE, 1994.

[8] T. F. Chan, J. Cong, T. Kong, and J. R. Shinnerl, Multilevel Optimization for large-scale circuit placement, *proc. IEEE International Conference on Computer Aided Design*, pp. 171-176, November 2000.