Performance Optimization on a Supercomputer with cTuning and the PGI compiler

Davide Del Vento

National Center for Atmospheric Research
Boulder, CO

EXADAPT 2012, London UK
3 March
About me

Davide Del Vento, PhD in Physics
Software Engineer, User Support Section
NCAR – CISL – Boulder, CO
http://www2.cisl.ucar.edu/uss/csg
http://www.linkedin.com/in/delvento
email: ddvento@ucar.edu
About NCAR

- National Center for Atmospheric Research
- Federally funded R&D center
- Service, research and education in the atmospheric and related sciences
- Various “Laboratories”: NESL, EOL, RAL
- Observational, theoretical, and numerical
- CISL is a world leader in supercomputing and cyberinfrastructure
Opinions, findings, conclusions, or recommendations expressed in this talk are mine and do not necessarily reflect the views of my employer.
Compiler's challenges

- Hardware is becoming more complex
- Some optimizations depend on frequently changing hw details
- Others are NP-complete
- Others are undecidable
- Hand-tuned heuristics are usually implemented in production compilers
- Other techniques provided better results
Need for speed

• Dramatic clock speed increase with Moore's law has stopped
• Science needs computation horsepower
• Hardware is becoming more complex
• Parallelism has become mainstream
• There is more interest in applying new research techniques to mainstream compilers.
Iterative compilation

- Compile a program with a set of different optimization flags
- Execute the binary
- Try again, until a satisfactory performance is achieved – of course this is a very long process
- ... and more
Predict optimization flags

- Use “somehow” the knowledge from iterative compilation, to find best optimizations quicker
- For example, pick flags with a strategy
- Note that the best optimization for a particular program on a particular architecture strongly depends on the program and the architecture
- Try Machine Learning
Existing cTuning CC infrastructure

- Feature extraction with MILEPOST GCC (56 features)
- Training infrastructure CCC (Continuous Collective Compilation) and cBench set of 20 training programs
- Machine Learning prediction infrastructure
- ... and more
Our contributions

- Implemented the PGI compiler in the framework
- Added a few benchmarks
- Reimplemented kNN
- Deployed on our system
PGI configuration file

1, 0, 4, -O
2, -fpic
2, -Mcache_align
3, 2, -Mnodse, -Mdse
3, 2, -Mnoautoinline, -Mautoinline
1, 20, 200, -Minline=size:
1, 5, 20, -Minline=levels:
2, -Minline=reshape
2, -Mipa=fast
3, 3, -Mnoire, -Mire=assoc, -Mnoire=noassoc
3, 2, -Mnomovnt, -Mmovnt
2, -Mnovintr
3, 3, -Mnopre, -Mpre, -Mpre=all
1, 1, 10, -Mprefetch=distance:
1, 1, 100, -Mprefetch=n:
3, 2, -Mnopropcond, -Mpropcond
2, -Mquad
3, 2, -Mnosmart, -Msmart
3, 2, -Mnostride0, -Mstride0
1, 2, 16, -Munroll=c:
1, 2, 16, -Munroll=n:
1, 2, 16, -Munroll=m:
3, 2, -Mvect=noaltcode, -Mvect=altcode
3, 2, -Mvect=noassoc, -Mvect=assoc
3, 2, -Mvect=nofuse, -Mvect=fuse
3, 2, -Mvect=nogather, -Mvect=gather
1, 1, 10, -Mvect=levels:num
2, -Mvect=partial
2, -Mvect=prefetch
3, 2, -Mvect=noshort, -Mvect=short
3, 2, -Mvect=nosse, -Mvect=sse
Training programs

<table>
<thead>
<tr>
<th>benchmark</th>
<th>suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. automotive_bitcount</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>2. automotive_qsort1</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>3. automotive_susan_c</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>4. automotive_susan_e</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>5. automotive_susan_s</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>6. bzip2e</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>7. network_dijkstra</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>8. office_stringsearch1</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>9. security_blowfish_d</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>10. telecom_CRC32</td>
<td>cBench [12]</td>
</tr>
<tr>
<td>13. 15.stream</td>
<td>HPCC [3]</td>
</tr>
<tr>
<td>15. 450.soplex</td>
<td>SPEC CFP2006 [8]</td>
</tr>
<tr>
<td>16. 999.specrand</td>
<td>SPEC [8]</td>
</tr>
<tr>
<td>17. adi</td>
<td>Livermore benchmarks kernel</td>
</tr>
<tr>
<td>18. liv14</td>
<td>Livermore loops</td>
</tr>
<tr>
<td>20. advect3d</td>
<td>kernel from COMMAS [5]</td>
</tr>
</tbody>
</table>
Deployment

- Reimplemented kNN in python
- Boring details of job submission and management on our machine
- Some glue from output of cTuning CCC to our data analysis, plots, etc
Iterative compilation
Convergence
Training

• The output of iterative compilation is fed to a machine learning algorithm

• In our case is simply kNN with k=1

• So the kNN learner is trained to select the “best” set of optimization flag, among the 20 sets (each for each example program)
Crossvalidation

• Leave-one-out crossvalidation is a commonly used technique to estimate ML

• Each training example is left out, the learner is retrained and used to predict the missing example

• It has a bias, but it is simple and still provides a useful evaluation so it is commonly used
Crossvalidation
Iterative compilation
A different look at the data (1)

- What can we learn from this result? How can we process it to learn more?
- Is the training set too limited?
- Do the features characterize correctly the example and instances (programs)?
- Are there too many features (kNN)?
- Could a different ML algorithm perform better?
A different look at the data (2)

- To answer these questions
- We ran an exhaustive search among the database of 19 “good” sets of optimization flags, for each leave-one-out program
- And selected the best
- This is the best that kNN can do for this dataset (e.g. changing or weighting the features)
Crossvalidation

![Crossvalidation Graph]

- Performance Ratio

1.2.blackscholes
1.4.fnform
1.5.HPCC-stream
1.2.clusterw
450.soplex
999.specrand
adi
advec3d
automotive_bitcount
automotive_qsort1
automotive_susan_c
automotive_susan_e
automotive_susan_s
bzip2e
liv14
network_dijkstra
office_stringsearch1
security_blowfish_d
telecom_CRC32

NCAR
Computational & Information Systems Laboratory
Upper limit to kNN cross-validation
First result

• Changing the way in which the distance is measured (e.g. removing irrelevant features) can improve performance
Upper limit to kNN cross-validation
Iterative compilation
More results (1)

- When exhaustive search is less performant than iterative compilation...
- Upper limit of kNN, regardless of distance evaluation is not competitive
- Adding more example programs might improve these cases
- Changing to an algorithm doing individual flag prediction (like SVN) might also improve these cases
More results (2)

- When exhaustive search is more performant than iterative compilation...
- We have discovered an important area of the optimization space not covered by iterative compilation
- Exploration of the optimization space with techniques different from the pure random space might find better results
Upper limit to kNN cross-validation
Iterative compilation
Conclusions

- We are interested in having an autotuning compiler deployed in production.
- We demonstrated that there is potential to improve performance, even of an already aggressively optimized compiler such as PGI.
- There is more work to do.
Aknowledgments

- NSF (National Science Foundation) for sponsoring NCAR and CISL
- CISL's internship program (SIParCS)
- Rich Loft, director of SIParCS and of a CISL's division, for his support to this work
- William Petzke and Santosh Sarangkar, 2011 interns of the SIParCS program for their contributions to this work.
Questions?