Eliminating PostgreSQL 19% cache-misses and 6% clock cycles by using GCC optimizations

Jim Tsung-Chun Lin 林宗俊
tclin914@gmail.com
December 5, PGConf Asia 2017 in Tokyo, Japan

Who am I?

- Jim Tsung-Chun Lin 林宗俊
- Compiler engineer in Andes Technology
- Interested in compiler optimization
- Focus on GCC and LLVM

Agenda

- Compiler
- Compiler Optimization
- Analysis and Profile on PostgreSQL
- Experiment Result
- Conclusion

Agenda

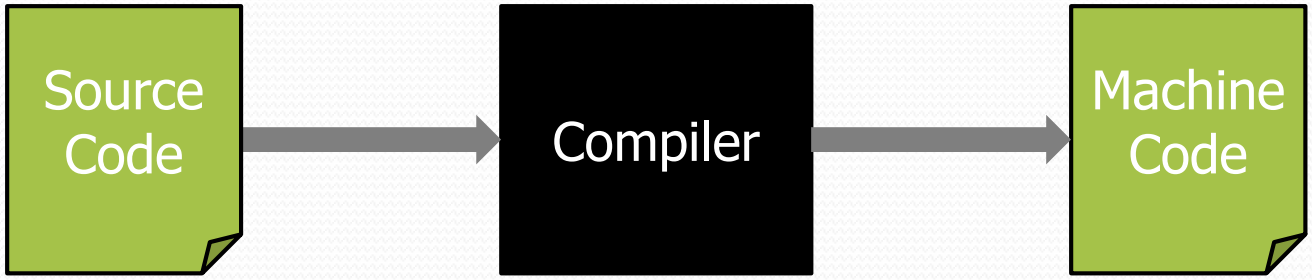
- Compiler
- Compiler Optimization
- Analysis and Profile on PostgreSQL
- Experiment Result
- Conclusion

Compiler

- Compiler translates information from one representation to another.
- Typically, compiler translates from high-level source code to low-level machine code
- The generated code must execute precisely the same computation as in the source code
- The quality of translated code is extremely importance

Compiler

- You can imagine that the compiler is like a black box.



GCC (GNU Compiler Collection)

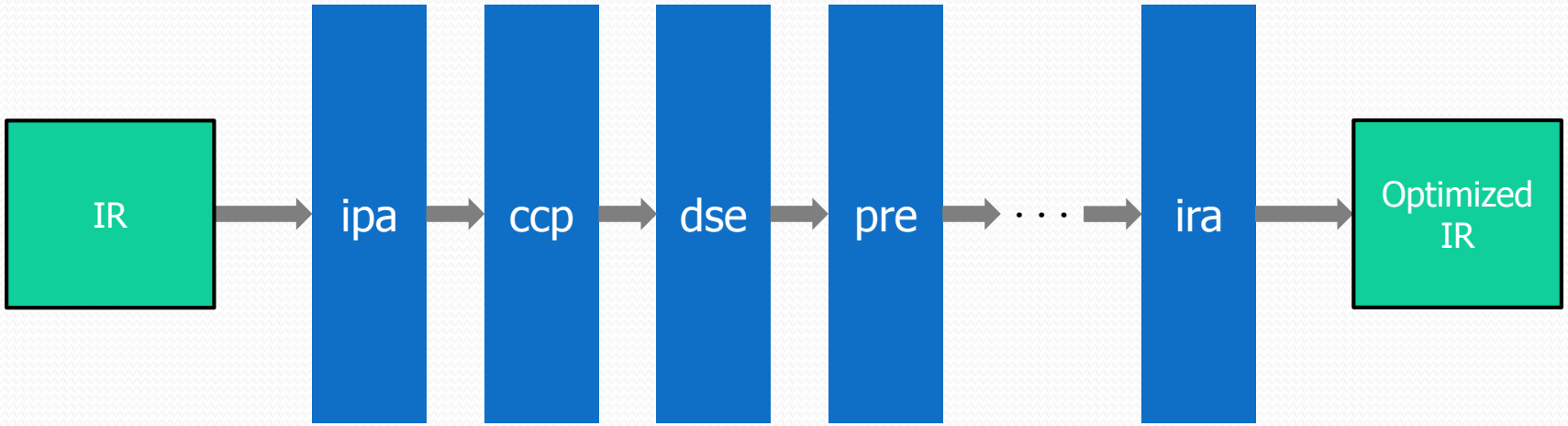
- GNU Compiler Collection (GCC) is a compiler system as part of GNU Project
- GCC is distributed under the GNU General Public License (GNU GPL)

Agenda

- Introduction
- Compiler
- Compiler Optimization
- Analysis and Profile on PostgreSQL
- Experiment Result
- Conclusion

Compiler Optimization

- Compiler optimizations = find *better* translations!
- Optimizations are made as a sequence of transformations.



Compiler Optimization – Dead Code Elimination

- Remove code which does not affect the program results.
- Benefit
  - Decrease code size
  - Reduce execution time

```
int bar() {
  int x = 2;
  int y = 1;
  int z = x + y;
  return x + y;
}
```

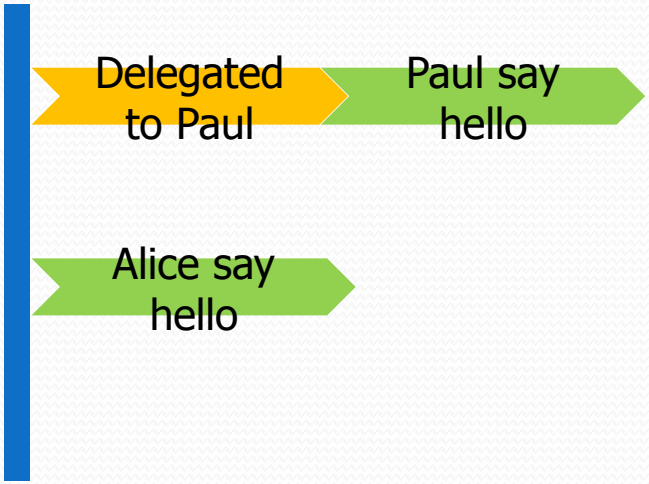
```
int bar() {
  int x = 2;
  int y = 1;
  return x + y;
}
```

Compiler Optimization - Inline Function

- Replace the function call with the function code itself.

```
void Alice() {
  Paul();
}

void Mary() {
  Paul();
}

void Paul() {
  printf("Hello!!!");
}
```

```
void Alice() {
  printf("Hello!!!");
}

void Mary() {
  Paul();
}

void Paul() {
  printf("Hello!!!");
}
```



save time for delegating others
= save time for making a function call

Compiler Optimization - Inline Function

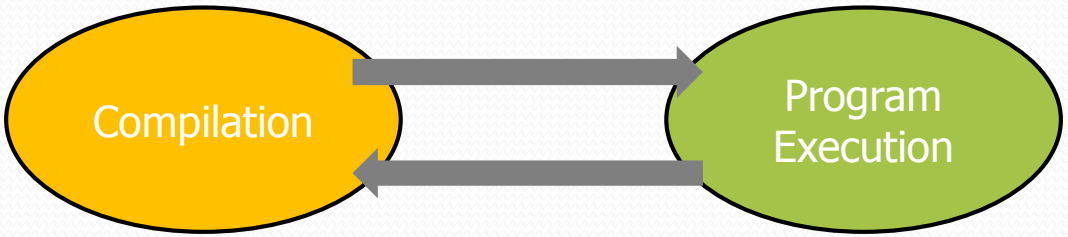
- Pros
  - Speed up your program by avoiding function calling overhead.
  - Increase locality of code cache reference
- Cons
  - Increase the executable size due to code expansion.
  - Larger executable size may cause more code cache-miss.

The Challenge of Compiler Optimization

- The optimizations made by compiler do not guarantee to get benefit in runtime.
- Moreover, some optimizations will result in other optimizations cannot be made or invalidated. Since optimization information may be removed from previous optimization pass.
- Depend on execution environment:
  - Operating System
  - Cache-size
  - CPU Pipelining
- To do or not to do optimizations is a big challenge.

Breaking the Compiler Optimization Restriction

- Use the runtime behavior of program to guide the compiler how to optimize.
- How to optimize = What kind of optimizations should be enabled or disabled ?
- Compilation -> Execution -> Compilation -> Execution …

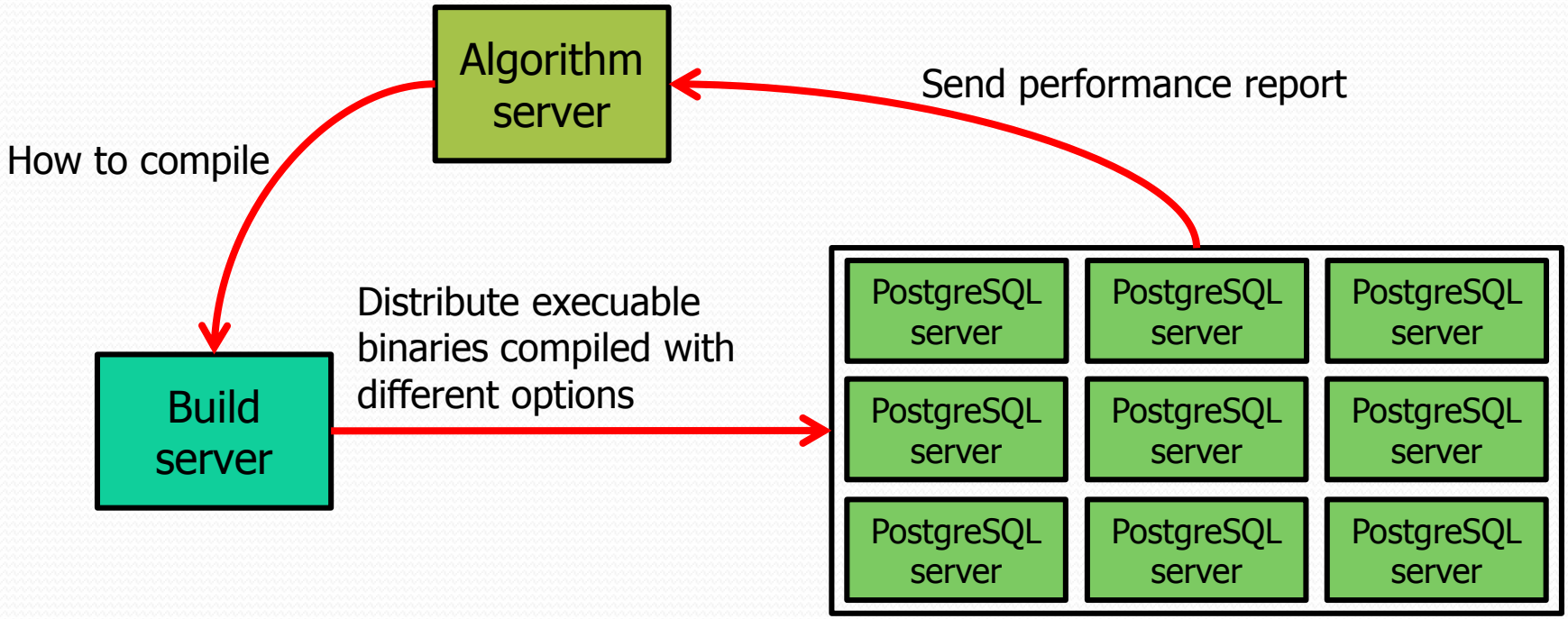


Another challenge

- GCC has a total of more than 100 optimization options, so the number of all possible combinations is a huge amount (more than 2^{100}).
- Compiling PostgreSQL source code into an executable binary takes about 2 minutes. If the compilation is 4000 times, it will take 55 days and have not yet included execution time.
- It is difficult or even impossible to run out of all possible combinations.

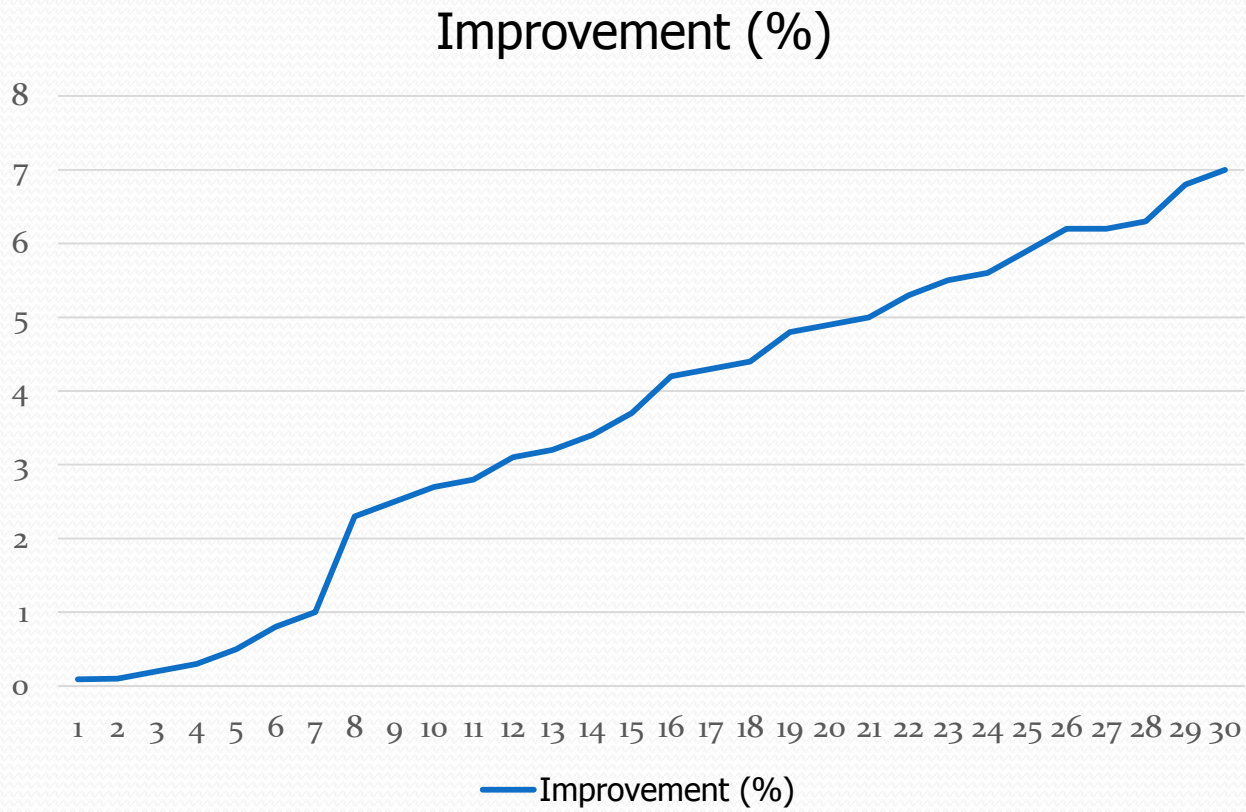
Applying Iterative Compilation on PostgreSQL

- Goal: Find a set of compilation options having better performance



Algorithm

- Quickly find compilation option having the best performance with machine learning.

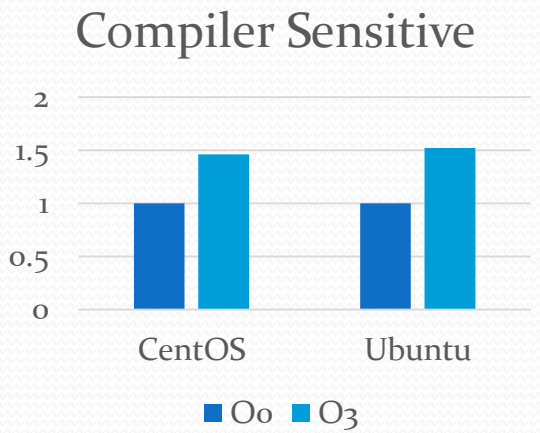


Benchmark

- The ratio of number of read and write operator in database system can reach up to 10^6 times at maximal level.
- That is, acceleration of select operator can significantly speed PostgreSQL up.
- pgbench
  - select-only

The Impact of Compiler Optimization on PostgreSQL

- A small compiler optimization experiments on PostgreSQL
  - -O0, without any optimization
  - -O3, with most of the optimizations
  - Performance of -O3 is about 1.5 times of -O0



- Performance of PostgreSQL will be affected by compiler optimization
- It is workable to improve performance by adjusting optimization options

Agenda

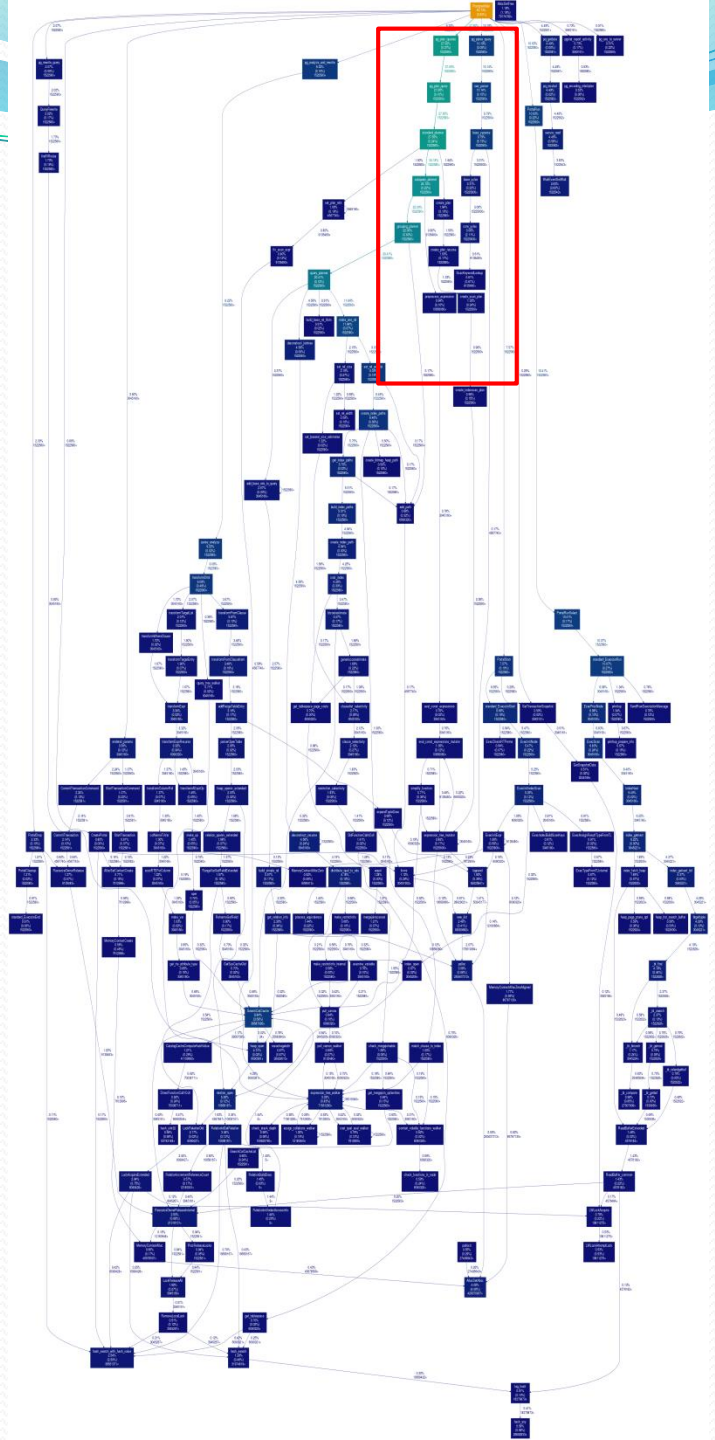
- Introduction
- Compiler
- Compiler Optimization
- Analysis and Profile on PostgreSQL
- Experiment Result
- Conclusion

Analysis of Function cycle ratio

Self	Command	Shared Object	Symbol
4.40%	postgres	postgres	AlloSetAlloc
3.66%	postgres	postgres	SearchCatCache
3.58%	postgres	postgres	base_yyparse
2.09%	postgres	postgres	hash_search_with_hash_value
1.17%	postgres	libc-2.17.so	__strcmp_sse42
1.15%	postgres	postgres	palloc
1.13%	postgres	postgres	MemoryContextAllocZeroAligned
0.98%	postgres	postgres	expression_tree_walker
0.94%	postgres	postgres	core_yylex
0.88%	postgres	postgres	LWLockAttemptLock
0.87%	postgres	libc-2.17.so	vfprintf
0.85%	postgres	libc-2.17.so	_int_malloc

Profile (gprof)

- Compiled with default options
- Deep and long function call chain



Analysis of Call Graph

- Total: The execution time from entering function to exiting function / the execution time of whole program
- Self: exclude the execution time of calling other functions

Function	total	self
PostgresMain	83.73%	0.80%
pg_plan_queries	27.92%	0.07%
pg_plan_query	27.80%	0.10%
standard_planner	27.65%	0.24%
subquery_planner	24.18%	0.22%
grouping_planner	22.00%	0.80%
query_planner	20.41%	0.12%

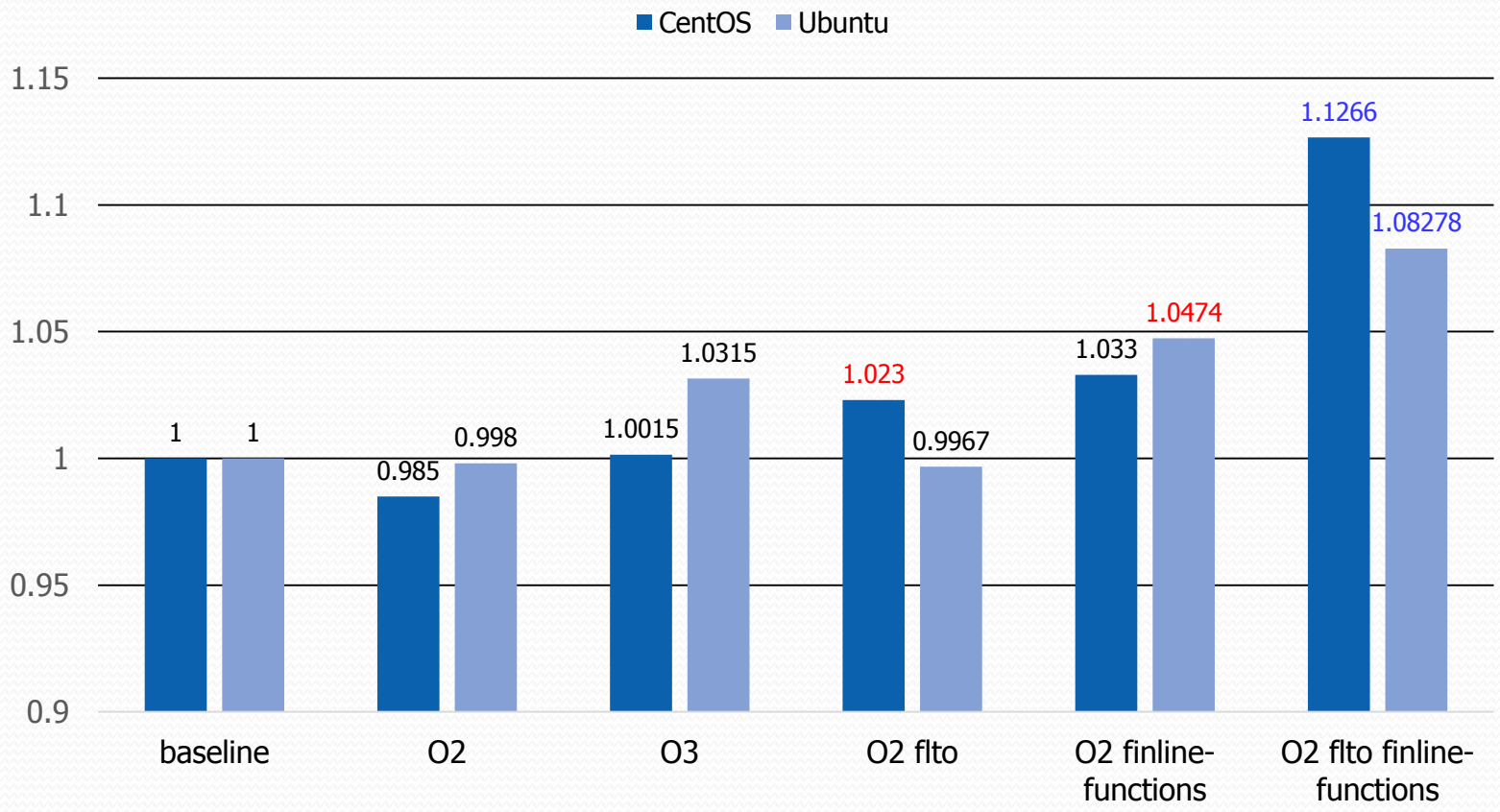
Agenda

- Compiler
- Compiler Optimization
- Analysis and Profile on PostgreSQL
- Experiment Result
- Conclusion

Experiment Result

- Definitions
  - TPS: Transaction Per Second
  - Latency: Time for one transaction
- Improvements
  - Increase TPS ↗
  - Decrease latency ↘
- Experiment
  - 8000 iterations
  - Take three days on eight machines

Experiment Result (tps)



Experiment Result (latency)

- Ubuntu

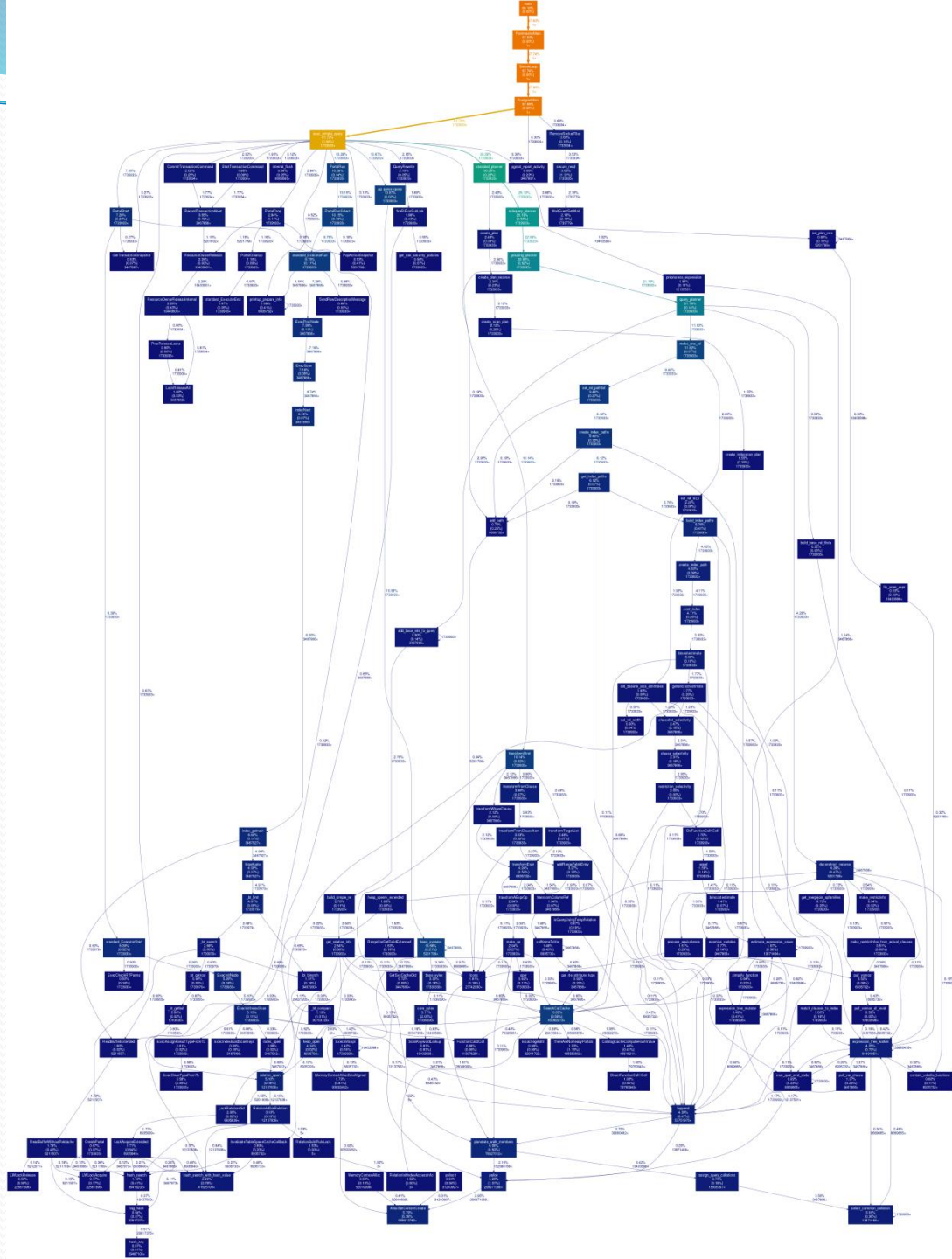
	Latency (ms)	Improvement (%)
baseline	0.044	0
-O2	0.049	-11.36
-O3	0.044	0
-O2 -flto	0.048	-9.09
-O2 –finline-functions	0.043	2.27
-O2 –flto –finline-functions	0.041	8.89

Experiment Result

- -finline-functions
  - Consider all functions for inlining, even if they are not declared inline.
  - Avoid function call overhead
- -flto (linking time optimization)
  - Merge all of object files as single optimizing unit
  - Optimization can work cross multiple object files
- If –flto and –finline-functions are enabled at the same time, then compiler can inline other object file's function.

Profile (gprof)

- Compiled with –O2 –flto –finline-functions options



Experiment Result

	Number of function	Reduction(%)
baseline	962	0
-O2	948	1.46
-O3	869	9.67
-O2 -flto	938	2.49
-O2 –finline-functions	882	8.32
-O2 –flto –finline-functions	728	24.32

Experiment Result

- Ubuntu

Number of #	Baseline	-O2 –flto –finline-functions	Improment (%)
Cache-miss	465616	376313	19.1795
Instruction	2093013	1975627	5.608
Clock Cycle	2128128	1990616	6.4616
IPC (Instruction Per Cycle)	0.983500	0.992470	0.912

- 19% cache-miss + 6% clock cycle = 8% improvement

Agenda

- Compiler
- Compiler Optimization
- Analysis and Profile on PostgreSQL
- Experiment Result
- Conclusion

Conclusion

- The runtime behavior of program can guides the compiler how to optimize program for getting better performance.
- Applying iterative compilation on PostgreSQL can achieve the best performance under different:
  - Hardware
  - Operating System
  - Scenario

Thank you