CALLGRAPH ANALYSIS & PERFORMANCE TOOLS DEVELOPMENTS

Roberto A. Vitillo
Lawrence Berkeley National Laboratory

ATLAS Software and Computing Workshop, 7 April 2011

COMPILER OPTIONS

- Compiler options that bring the greatest benefits[1] are the ones that permit to:
  - don’t keep the frame pointer in a register for functions that don’t need one
  - inline functions
- Enable Streaming SIMD Extensions (SSE)
  - Use SSE for scalar floating point math: still some alignment issues with gcc-4.3.5, solved in gcc-4.4.4
  - Use SSE vector instructions were possible (see gcc built-in functions)
  - Use glibc with SSE support (glibc >= 2.10 with IFUNC), e.g.: improved memcpy, memmove
  - Enable autovectorization

[1] KapilVaswani, P. J. Joseph, Matthew J. Thazhuthaveetil, and Y. N. Srikant. 2007. Microarchitecture sensitive empirical models for compiler optimizations. In Proceedings of the International Symposium on Code Generation and Optimization

CALLGRAPH ANALYSIS

- Problem:
    - low instruction retired / call retired ratio
    - high call retired / branch retired ratio
- Inlining functions called millions of times per event can indeed bring considerable benefits, e.g.:
    - *Trigger/TrigT1/TrigT1RPChardware* - 4% instruction retired reduction (*RecExCommon/bstoesd, 1 function inlined*)
    - *TileCalorimeter/TileCalib/TileCalibBlobObjs* - 5% instruction retired reduction (*RecExCommon/bstoesd, 5 functions inlined*)

CALLGRAPH ANALYSIS

- David Levinthal's proposal:
  - "Use Last Branch Records (LBR) and static analysis to evaluate frequency and cost of function calls"
  - "Use social network analysis / network theory to identify clusters of active, costly function call activity"
  - "Order cluster by total cost and inline"

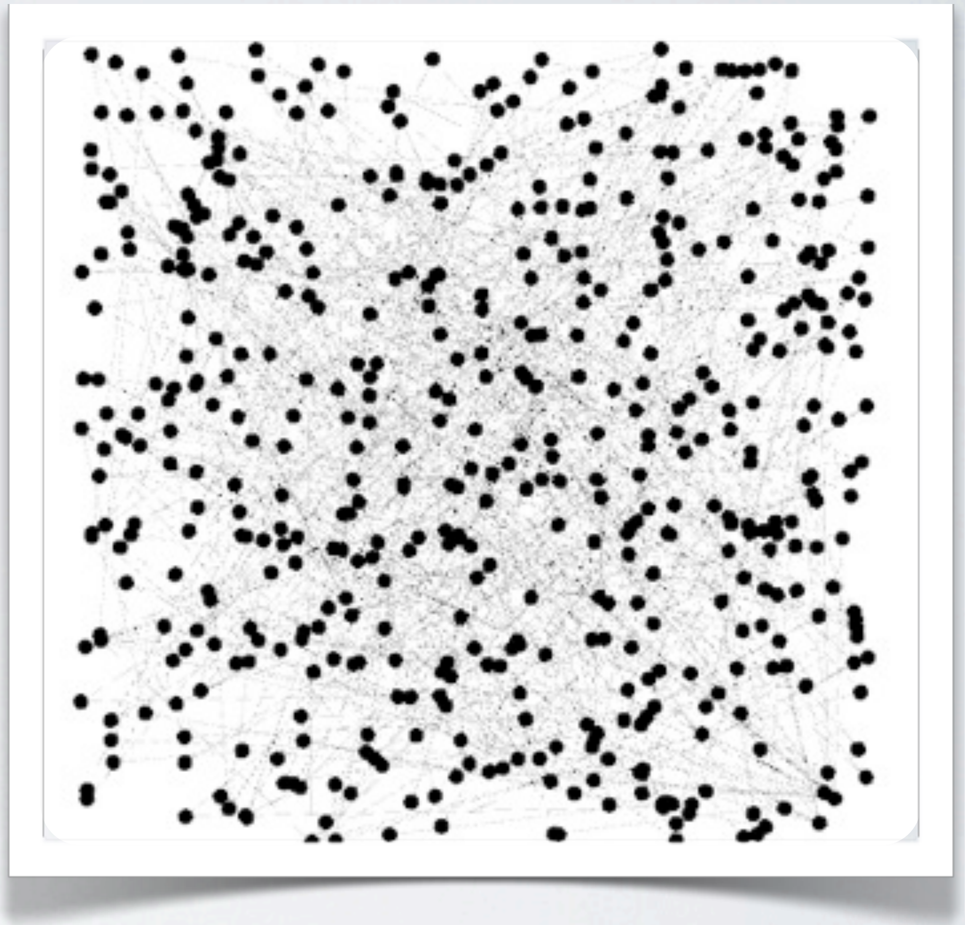
CALLGRAPH ANALYSIS

- Callgraph of a five event reconstruction job (d482) built from callgrind output (40 KN, ~160 KE)

- For visualization purposes we consider only the following sub-callgraph (~500 N, ~600 E)

  - Nodes > 0.5% total executed instructions

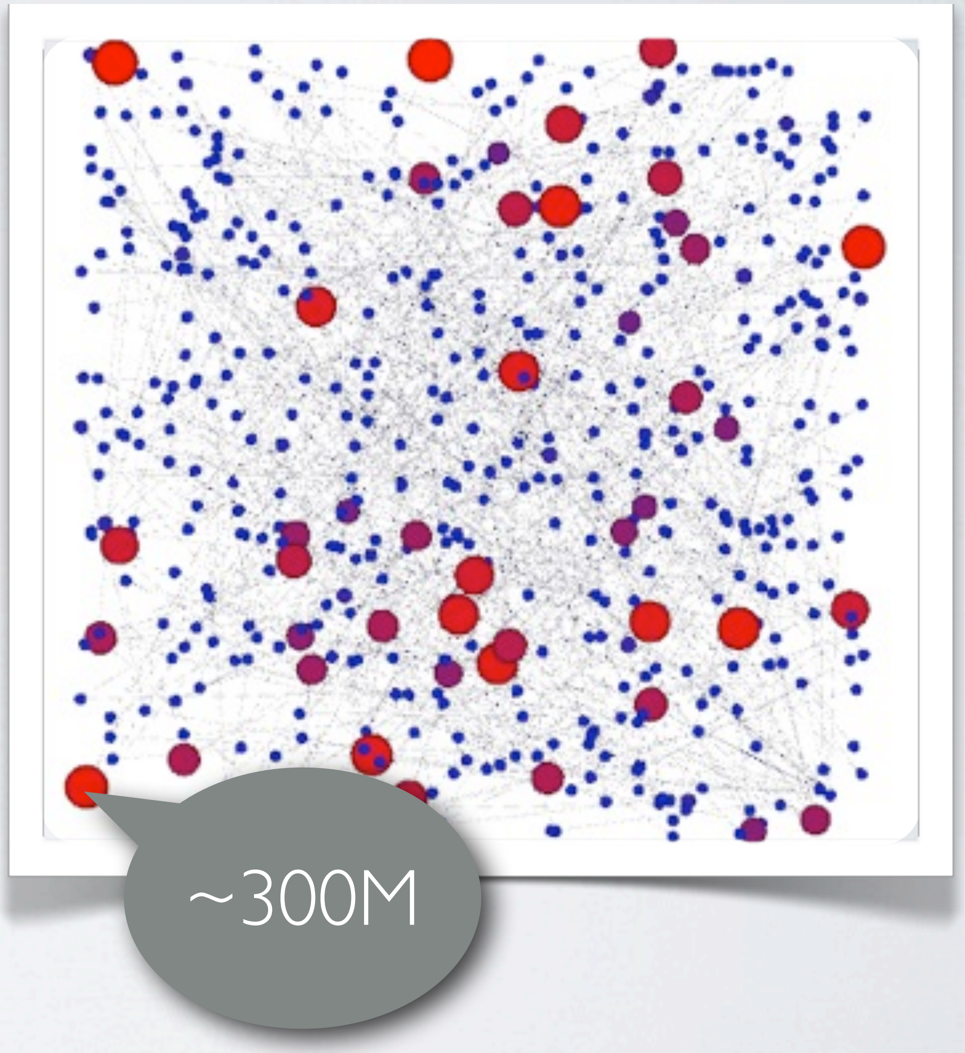
  - Arcs > 0.1% relative frequency



CALLGRAPH ANALYSIS

- Nodes with higher weighted degree (WD) are highlighted with a “warmer” color and have a bigger size

- Where are the cluster? Naive approach:
    - Build a new subgraph containing only nodes with WD > threshold and their respective edges
    - Find the connected components and compute the cost for each of them
    - Inline the clusters by descending cost



CALLGRAPH ANALYSIS

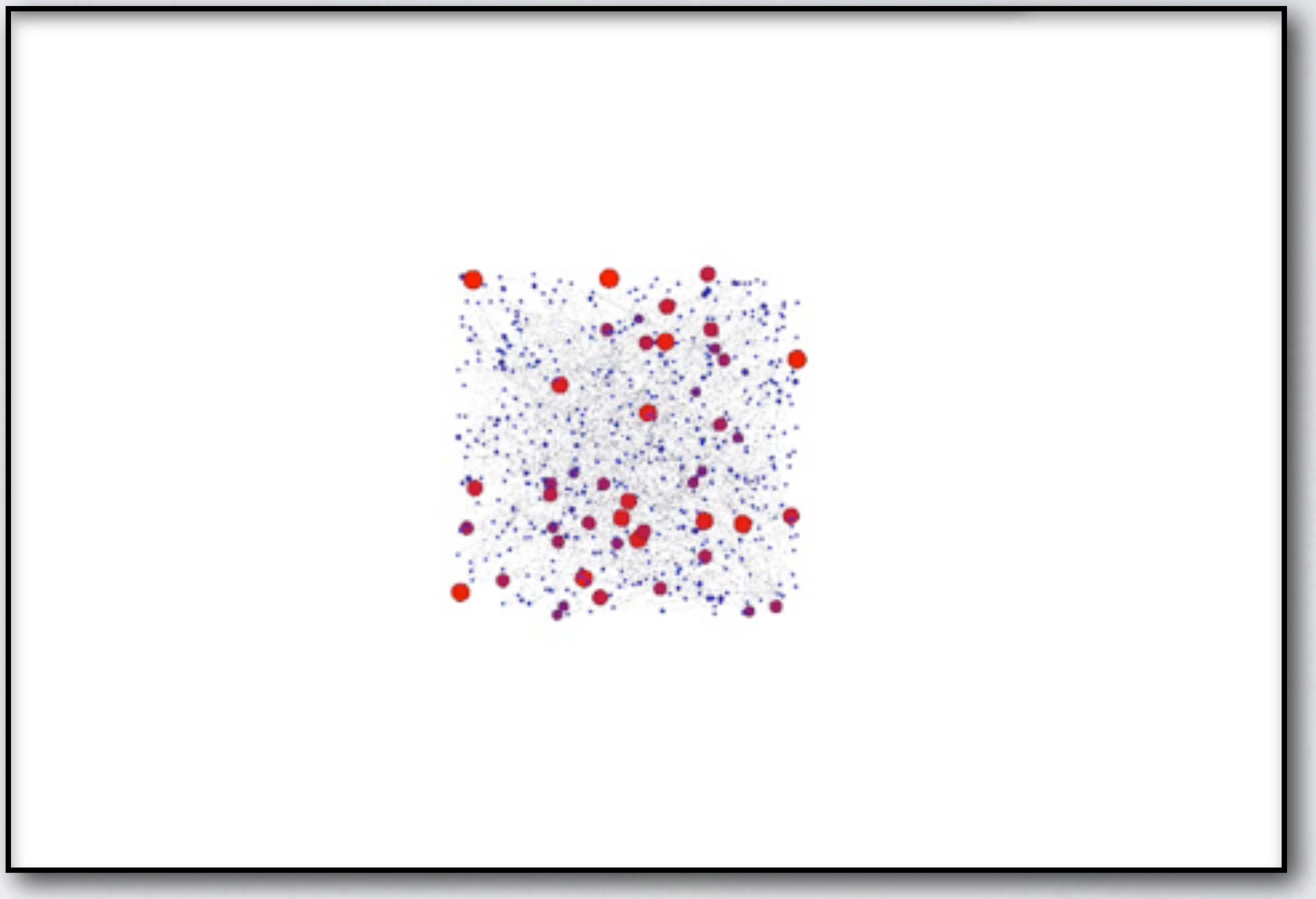
- Use of a force based algorithm to layout the graph and visualize the clusters
  - Nodes act as point charges
  - Arcs act as springs

layout animation

- Callgraph cluster analysis and inlining could be embedded in a compiler through the PGO component

CALLGRAPH ANALYSIS

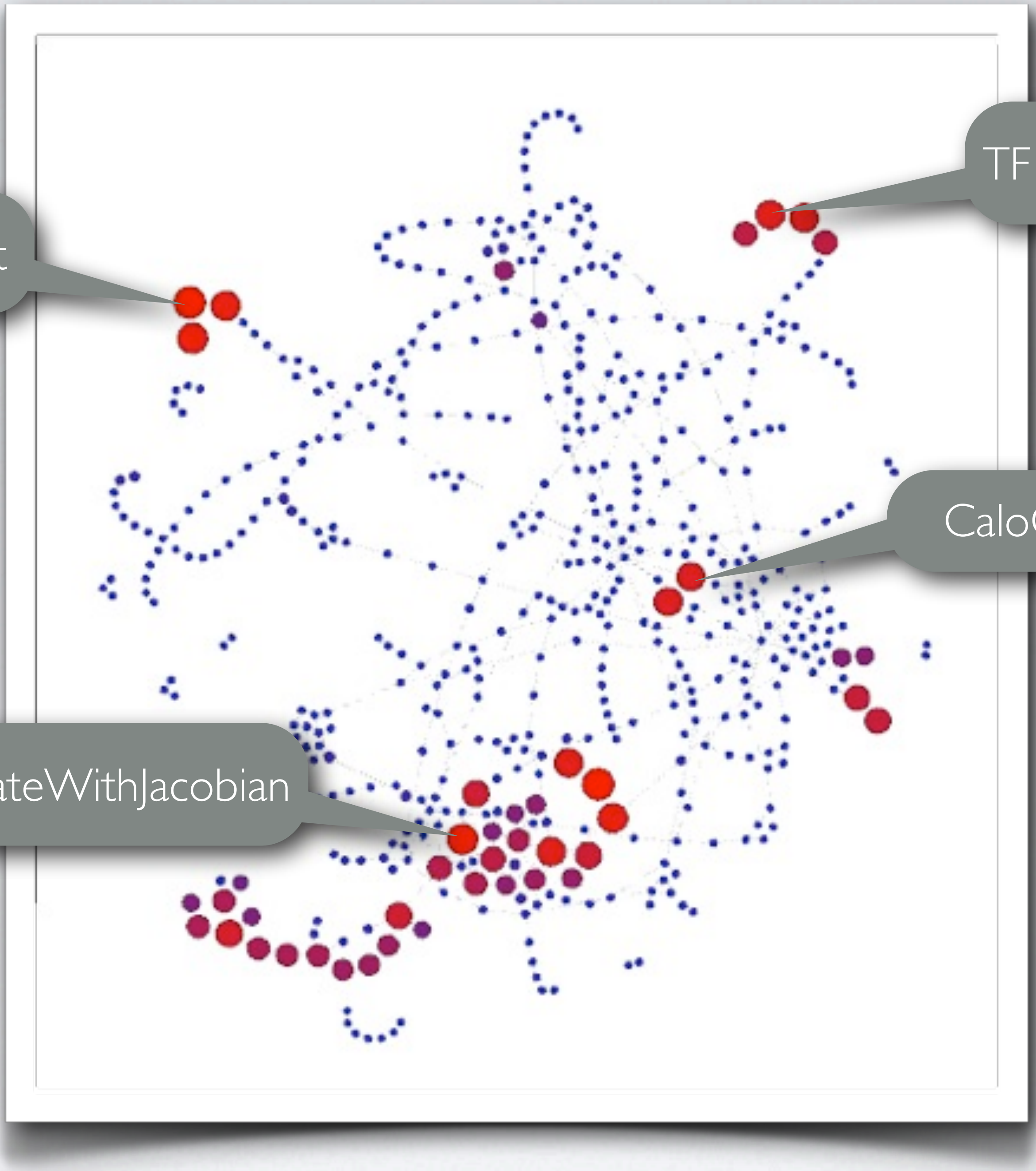
- Use of a force based algorithm to layout the graph and visualize the clusters
  - Nodes act as point charges
  - Arcs act as springs



layout animation

- Callgraph cluster analysis and inlining could be embedded in a compiler through the PGO component

CALLGRAPH ANALYSIS



TF1_EvalWrapper::DoEval

std::Rb_tree_increment

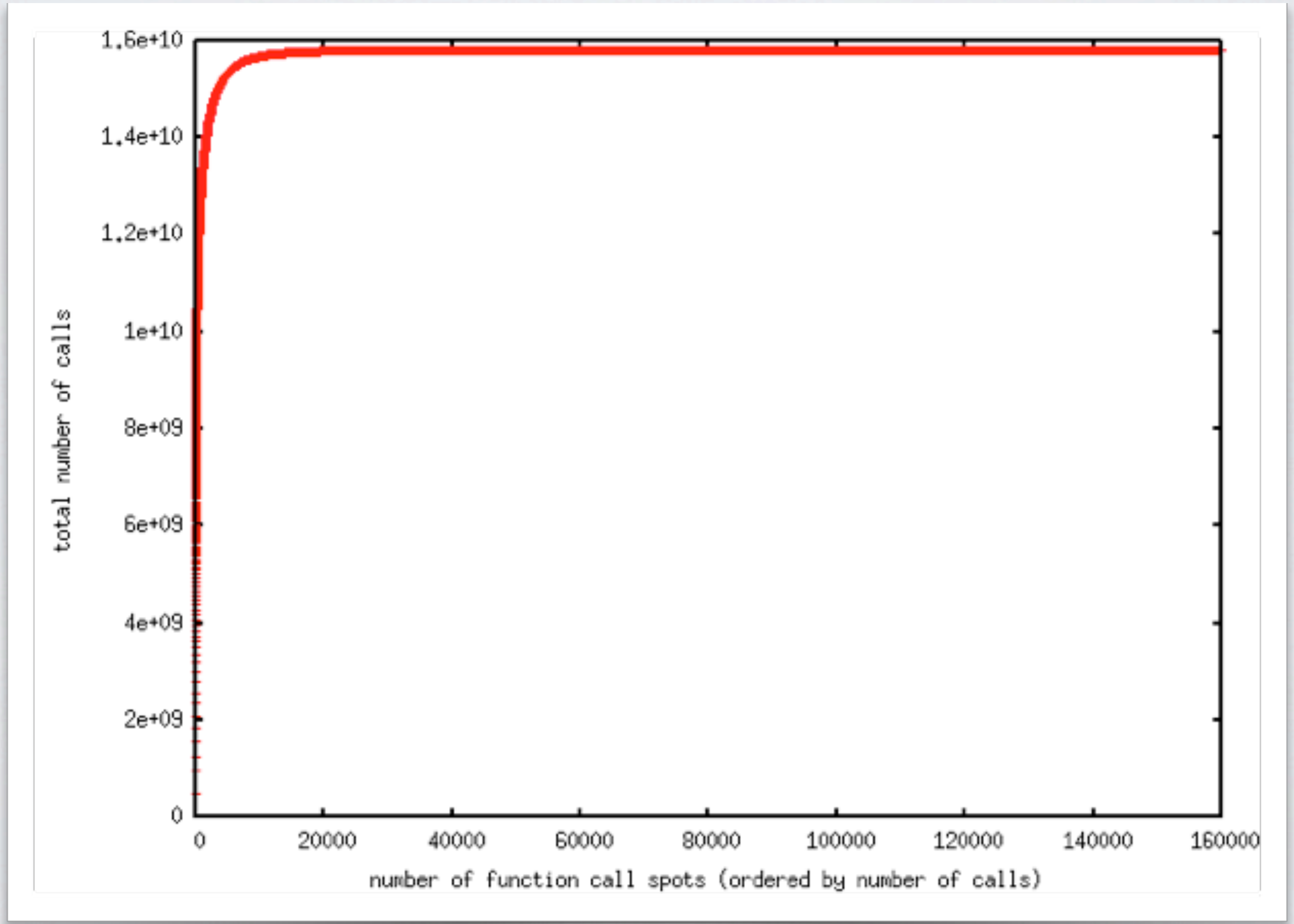
CaloCluster::eta

Trk::RungeKuttaPropagator::propagateWithJacobian

Also notice the chains: usually a function calls only one “heavy” function

CALLGRAPH ANALYSIS

- More clusters in the full graph: ~1% (2K/160K) of call spots make up for ~90% (14.2G/15.8G) of all function calls!
- Complete sorted list available on lxplus: *~vitillo/public/callspots*



CALLGRAPH ANALYSIS

- Not every function can be inlined:
  - Third party library functions: use Link Time Optimization (LTO) + Profile Guided Optimization (PGO) if possible (LTO needs static libraries and a recent compiler)
  - Virtual functions: use explicit qualification or final keyword (c++0x or custom patch) where possible & compiler devirtualization support if available
- Inline functions only in specific spots: use alternative versions; introduce a pragma in combination with LTO; use LTO + PGO
- Conclusion: To try to solve the problem automatically we need LTO (gcc-4.6) and some form of PGO

WHAT NEXT?

- Problem: High indirect call / call retired ratio
- Possible solution: don’t use position independent code
- Will the performance gain be greater than the amount of unsharable library code?
- On x86-64 PIC is mandatory for shared libraries
- At this point, should we consider to use static libraries that can be used also for LTO?

PERFORMANCE TOOLS

- **IgProf**
  - simple tool for measuring and analyzing application memory and performance characteristics
  - no changes to the application or build process required
  - fix for Athena developed
- **Systemtap**
  - useful to "dynamically instrument" specific functions and much more
  - provides a simple command line interface and scripting language for writing instrumentation
  - with uprobe kernel module it can be used also with userspace code
- **Perf-events**
  - Kernel module that permit to access sw and hw performance counters

PERFORMANCE TOOLS

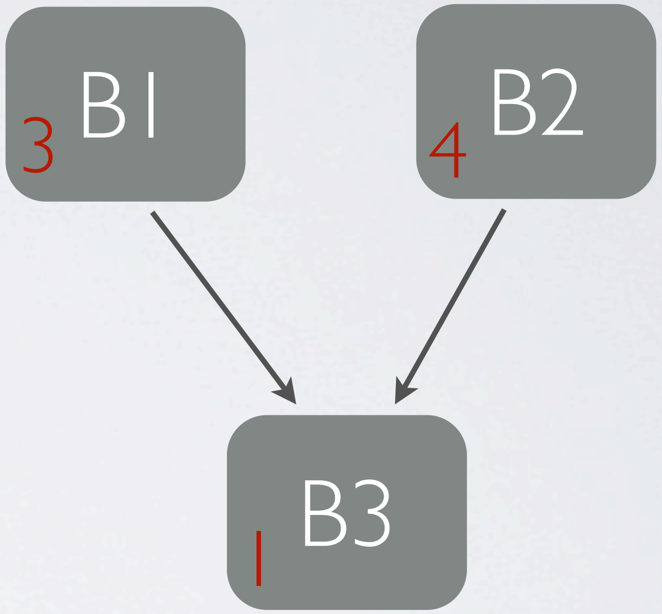
- Collaboration with Google: David Levinthal & Stephane Eranian
- Short term goal: use kcachegrind to visualize perf-events reports
    - Benefits: performance wise an order of magnitude faster than using instrumented code
- Long term goal: develop an open source visualizer that shows collected data with emphasis on OO applications

PERFORMANCE TOOLS

- LBR is used to evaluate the frequency of function calls
- Sampling performed on the *BR_RET_EXEC* events (available on Sandy Bridge architectures)
  - *BR_CALL_EXEC* cannot be used directly with trampolines
- Caveat: LBR is currently not available on the user side of perf-events
  - Kernel patch to dump the LBR is ready - filtering still missing
  - The patch cannot be accepted until there is an useful use-case integrated within the tools
  - Proposed simple use-case: % of branches inside and outside of a module
- Random sampling has been added to the kernel to avoid synchronization issues

PERFORMANCE TOOLS

- Improve basic block counts by:
  - using branch records to generate software instruction retired event
  - adhering to flow conservation rules while limiting the amount of changes to sample counts to a minimum



In general with sampling #B1 + #B2 != #B3

CONCLUSIONS

- We knew that there was a problem and now we know what to fix in order to solve it

- Inlining of the clusters will require some manual changes to the codebase but a more general solution needs LTO & PGO

- Avoid PIC?

- Next generation performance tools with emphasis to OO software needed