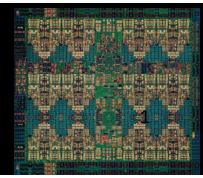
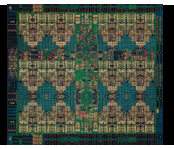
Application Performance Tuning on POWER9 with Opensource compilers

Dr. Archana Ravindar (aravind5@in.ibm.com)

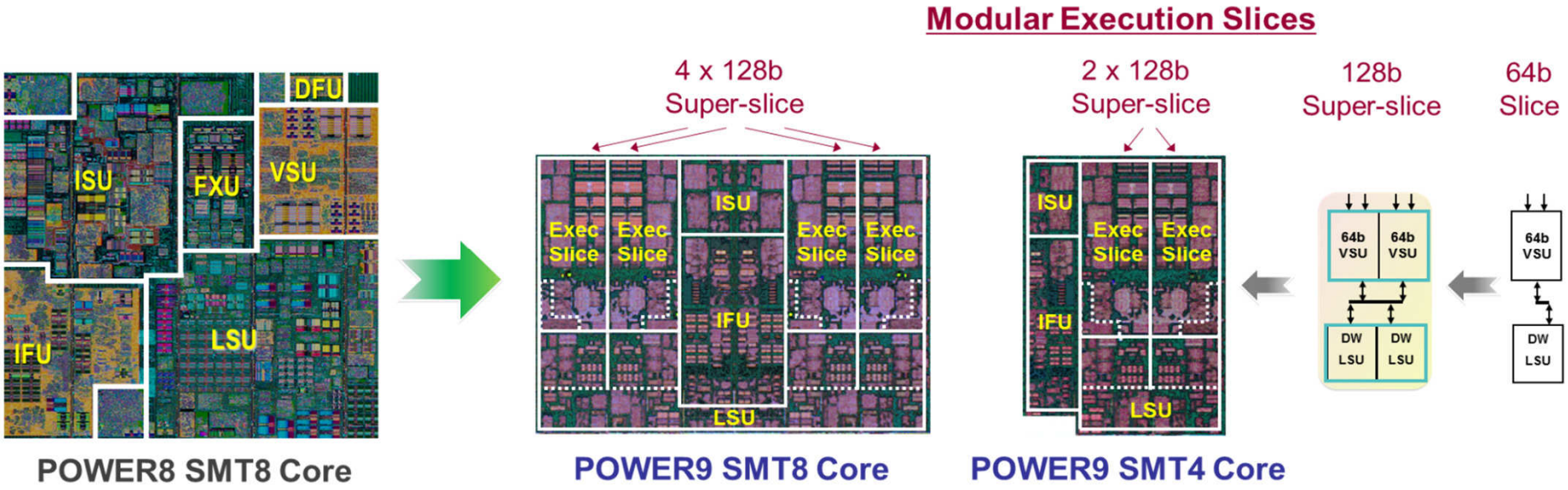


Scope of the Presentation

- Outline Tuning strategies to improve performance of programs on POWER9 processors
- The strategy is directed by program characteristics which can be assessed by hardware performance counters
- These strategies can take the form of compiler flags, source code pragmas/attributes
- This talk addresses overall summary of options supported by open source compilers such as GCC, LLVM in comparison with IBM XL
- Tools used to measure performance counters- perf / PAPI



POWER9 Processor

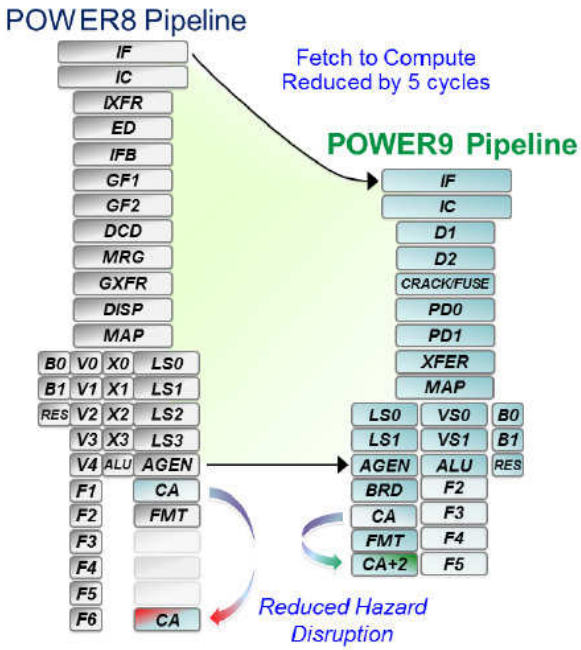


Reference: IBM Power9 Processor Architecture, S. Sadasivam, et al, IEEE Micro, Volume 37, Issue : 2, Mar-Apr 17

- Refactored Core provides Improved Efficiency and Workload Alignment
- Enhanced Pipeline efficiency with modular execution and intelligent pipeline control
- Increase pipeline utilization with symmetric data type engines : Fixed, Float, 128b, SIMD
- Shared compute resource optimizes data type interchange

POWER9 Core Pipeline Efficiency

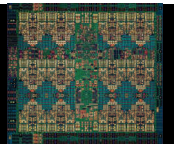
- Shorter Pipelines with reduced disruption
- Improved Application Performance for Modern Codes
- Advanced Branch Prediction
- Higher Performance and Pipeline Utilization
  - Removed instruction grouping
  - Enhanced instruction fusion
  - Pipeline can complete upto 128 (64-SMT4) instructions /cycle
- Reduced Latency and Improved Scalability
  - Improved pipe control of load/store instructions
  - Improved hazard avoidance



Reference: IBM Power9 Processor Architecture, S. Sadasivam, et al, IEEE Micro, Volume 37, Issue : 2, Mar-Apr 17

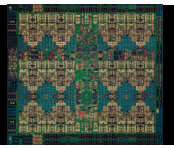
Tools that we use in the Discussion

- Open source compilers such as GCC; We also discuss available options in other compilers such as LLVM, Clang on POWER
  - [gcc| clang] -O[n] program.c –o program for C programs
  - [g++| clang++] -O[n] program.cc –o program for C++ programs
  - Optimization level ranges from 0 to 3, Ofast
  - -mcpu=power9 for targeted code generation on POWER9
  - Profile directed feedback (-fprofile-generate and –fprofile-use)
- Perf tool
  - To record hotspots/profile application
    - perf record -e r<code> ./binary args > out (produces perf.data)
    - perf report (opens profile report stored in perf.data)
  - To measure hardware events
    - perf stat –e r<code> ./binary args > out
  - These counters can be read using PAPI API as discussed in the previous session



Performance Opportunities in the Front-End

- POWER9 is a superscalar processor and is pipeline based
- A region of straight line code zooms away across the pipeline to completion whereas branches introduce stalls as the processor now has to compute a condition to decide which direction to fetch the next instruction from
- Branch prediction plays a critical role in determining the performance
  - Misprediction causes wrong path instructions to be fetched and introduces additional penalty as these instructions need to be flushed from the pipeline and correct instructions need to be fetched and processed
  - Counters to detect this: PM_BR_MPRED*
- Branches are caused even by function calls, Such branches affect instruction cache locality and increase instruction cache misses
  - Counters to detect this: PM_L1_ICACHE_MISS
- Branches within loops hinder vectorization opportunities

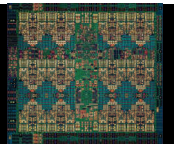


Tuning Strategies to Improve Performance in the Front-End

- Unrolling loops (will reduce loop branches and in some cases branches within loop)
- Example of unrolling
- Enforcing unrolling in source
- Place #pragma unroll(N) before the for loop which needs to be unrolled
- Compiler support for controlling Unrolling
  - Enable Loop Unrolling: -funroll-loops: Leave it to the compilers judgement to decide optimal unrolling for each loop
  - Disable Loop unrolling : -fno-unroll-loops

```
int x;

for(x=0;x<100;x++
  ) {

    func(x);

}
```

```
int x;

for(x=0;x<100;x+=5)
  {

    func(x);

    func(x+1);

    func(x+2);

    func(x+3);

    func(x+4);

  }
```

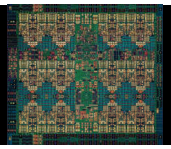
Tuning Strategies to Improve Performance in the Front-End

- One line Branches / If-Conversion
  - Wherever possible, simplify complex branches into one line branches as shown
- Compiler generates isel instructions for such branches that essentially converts a control dependency into a data dependency
- GCC/LLVM option to generate isel : -misel
- To Disable generation of isel: -mnoisel

```
If(val==M)
 a=b; else
 a=c;
```

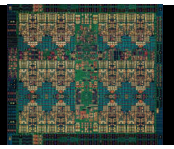
```
a=(val==M)?b:c;
```

– Other techniques to improve performance: Provide hints in source code to indicate the expected values of expressions appearing in branch conditions (long __builtin_expect(long expression, long value);) *(hint whether branch is more likely to be taken/not)*



Tuning Strategies to Improve Performance in the Front-End

- Inlining routines (will reduce branches due to function call jump/return)
- Will also help in better scheduling but inlining always is not a good idea. Compiler is the best judge
- Enforcing inlining in source – use inline __attribute__((always_inline)) in front of a function definition
- Compiler Support
  - -finline-functions : Inline suitable functions
  - -fnoline-functions: Disable inlining
- Help Compiler to Inline more Functions:
  - Avoid function pointers/Indirect calls, virtual calls as much as possible
    - Compiler cannot figure out statically, which function is going to be called from a call site
    - Such functions are difficult to be inlined by the compiler

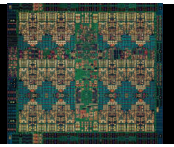


Tuning to Improve Performance of Load Store Unit

- In a RISC architecture, predominantly, instructions operate on registers
- Load, store instructions used to transfer data from memory to registers
- When #live variables > #available registers, spill is performed
- 1 spill = 1 store + 1 load

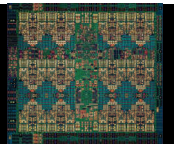
- *Spilling hot variables can hit performance*
- Spills increase Path length, address arithmetic instructions
- Unnecessary reads/writes to memory

- Issues due to spills detected in following counters-PM_LSU_FIN, PM_LSU_FLUSH, PM_LSU_REJECT_LHS , PM_INST_CMPL,



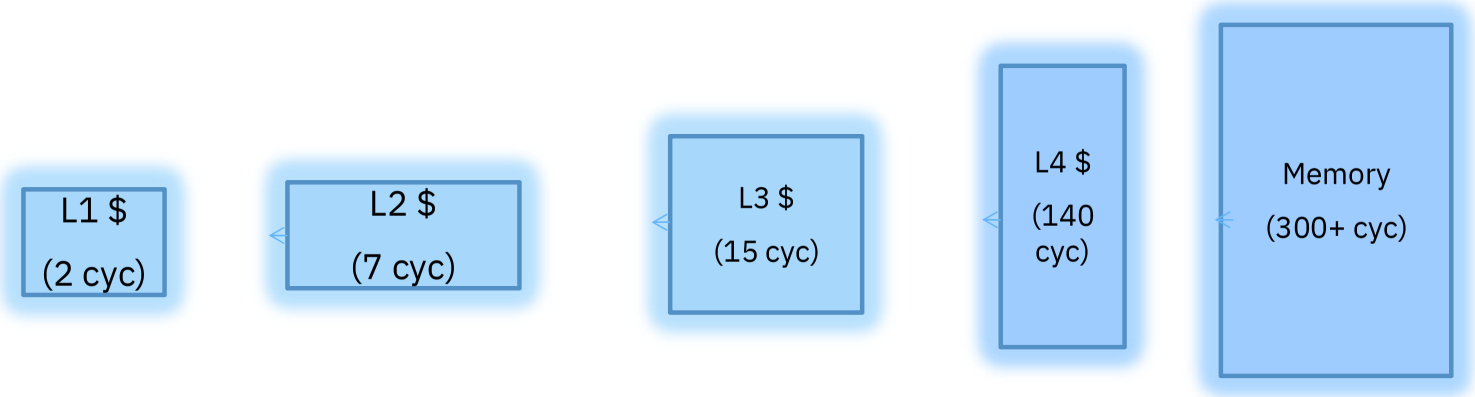
Tuning to Improve Performance of Load Store Unit

- Limit extensive unrolling/inlining that can cause long-live ranges of variables
- Use other register resources like SIMD registers if applicable (Vectorization)
  - GCC/Clang Options -mvsx, -maltivec, -ftree-loop-vectorize, Clang (-mllvm –force-vector-width=n) forces generation of SIMD instructions automatically
- POWER9 has introduced several new instructions in various domains that can be used to reduce pressure on registers
  - Array indexing improvements.
  - String operation support (e.g., for strcpy, strcmp).
  - Character operation support (e.g., for is_alpha).
  - Integer remainder (mod).
  - 64-bit integer multiply-add (for larger data types).
  - Count trailing zeros.
- Inorder to generate new ISA automatically, we need to compile the program with the option –mcpu=power9
- Using –mtune=power9 generates code that is scheduled optimally for the POWER9 core



Tuning to Improve Performance of Load Store Unit

- Memory is organized in a hierarchy;
- L1 cache: Closest memory to the processor and the fastest, followed by L2, L3 upto main memory



- Memory is most distant to the processor and slowest
- Data cache: stores data, instruction cache: stores instructions
- Data cache misses can stall load instructions in the pipeline causing a cascading effect on all those instructions dependent on it
- Counters that can help detect performance issues due to memory -PM_LD_MISS_L1, PM_CMPLU_STALL_DCACHE_MISS, PM_CMPLU_STALL_DMISS_LMEM etc

Tuning to Improve Performance of Load Store Unit

- Hardware prefetching
  - Controlled by DSCR (data stream control register) settings;
    - –ppc64_cpu --dscr=<n>
  - Common DSCR configurations: n=0 (moderate depth, ramp) : By default the HW prefetcher is “ON”
  - n=0x1D7 (Achieve most aggressive depth, most quickly, enable stride N prefetch),
  - n=0x1 (no prefetch)
  - Reference: https://developer.ibm.com/linuxonpower/docs/linux-on-power-application-tuning/

- Software prefetching
  - Programmer inserted prefetch instructions __dcbt (load prefetch), __dcbtst (store prefetch)
  - If you want to explicitly control prefetching via software *only*, you can turn off hardware prefetching using ppc64_cpu –dscr=1

Tuning to Improve Performance of Load Store Unit

- XL provides an option to enable compiler to insert prefetch instructions wherever applicable (-qprefetch)
  - -qprefetch=aggressive
  - -qprefetch=noaggressive
  - Disable prefetch (-qnoprefetch)

POWER9 supports setting DSCR values at compile time for an application

Example: -qprefetch=aggressive:dscr=<value>

Similar GCC prefetch option: -fprefetch-loop-arrays/-fno-prefetch-loop-arrays

Tuning to Improve Performance of Arithmetic Unit

- Compute intensive programs whose computations can be parallelized can take advantage of vector instructions on POWER
  - Advantages- reduces loads, stores and hence pathlength, reduces register pressure on GPRs, effective use of resources, Faster throughput
- At a time- Vector instructions can work on 4 32 bit words, 8 half-words and 16 bytes
  - Amount of work done per unit time correspondingly becomes faster
- Clang/GCC: -ftree-loop-vectorize, -mvsx, -maltivec, -mllvm –force-vector-width=n(Clang only)
- Help the Compiler to automatically vectorize loops
- Keep the loop simple
- Avoid extensive branches, pointer references within loops (use restrict wherever applicable)
- GPU codes can scale really well with SIMDization for performance
  - Structure of arrays are more amenable to vectorization than array of structures

Additional Ways to Improve Performance

- Serial v/s Parallel Execution
  - If there are multiple tasks which do not have a dependency amongst each other and can be done in parallel use a framework such as OpenMP that can perform Tasks in 1/Nth time with N threads
- If Mathematical accuracy is not important, use -Ofast
  - This automatically substitutes expensive library calls to native implementation of the math function using target ISA
- Thread Binding
  - We can use GOMP_CPU_AFFINITY="0 8 16 24 32 40 48 56" OMP_NUM_THREADS=8 time ./application <params> to bind first thread to CPU0, second thread to CPU8, ... so on
  - The ordering of CPU numbers determines performance of the application
  - If all threads are bound to a single CPU execution speed slows down
  - To choose the right CPU number on a POWER Linux system, we can consult the file /sys/devices/system/cpu/cpu0/topology/thread_siblings_list

Comparison of Common Flags – LLVM/GCC/XL on POWER9

Flag Kind	XL	GCC/LLVM	Equivalent in source	Benefit	Drawbacks
Unrolling	-qunroll	-funroll-loops	#pragma unroll(N)	Unrolls loops ; increases opportunities pertaining to scheduling for compiler	Increases register pressure
Inlining	-qinline=auto:level=N	-finline-functions	Inline always attribute	increases opportunities for scheduling; Reduces branches and loads/stores	Increases register pressure; increases code size
Enum small	-qenum=small	-fshort-enums	-	Reduces memory footprint	Can cause issues in alignment
isel instructions		-misel		generates isel instruction instead of branch; reduces pressure on branch predictor unit	latency of isel is a bit higher; Use if branches are not predictable easily
General tuning	-qarch=pwr9”, -qtune=pwr9"	-mcpu=power8, -mtune=power9			
64bit compilation	-q64	-m64			
Prefetching	-qprefetch	-fprefetch-loop-arrays	__dcbt/__dcbtst, _builtin_prefetch	reduces cache misses	Can increase memory traffic particularly if prefetched values are not used
Link time optimization	-qipo	-flto , -flto=thin		Enables Interprocedural optimizations	Can increase overall compilation time
Profile directed feedback	-qpdf1, -qpdf2	-fprofile-generate and –fprofile-use LLVM has an intermediate step llvm-profdata		Enables hot path optimizations	Requires a training run

Summary

- Today we talked about
  - Tuning strategies pertaining to the various units in the POWER9 HW –
    - Front-end, Load Store unit, Arithmetic Unit
    - Some of these strategies were compiler flags, source code pragmas that one can apply to see improved performance of their programs
  - We also saw additional ways of improving performance such as parallelization, binding etc
  - We saw that POWER9 has the most comprehensive set of hardware counters that enable analysts to understand applications of performance and get to the bottlenecks quickly
    - Get counter data either using perf stat or PAPI APIs
  - We concluded with a comparison of compiler flags on open source compilers such as GCC, LLVM with IBM XL compilers

Disclaimer: This presentation is intended to represent the views of the author rather than IBM and the recommended solutions are not guaranteed on sub optimal conditions