Architecture Cloning For PowerPC Processors

Edwin Chan, Raul Silvera, Roch Archambault
edwinc@ca.ibm.com
IBM Toronto Lab

Oct 17th, 2005

Outline

- Motivation
- Implementation Details
- Results

Scenario

Previously, only 2 solutions exist for the IBM XL Compiler to create an executable compatible with multiple PowerPC processors:

- *Generate generic instructions*
  - Unable to take advantage of the latest hardware features
  - Suboptimal performance on all platforms

- *Recompile the application for different architectures*
  - Recompilation takes a long time
  - Adds building complexity, more support headaches, longer time to ship

- *Example: ISV (Independent Software Vendor)*

Our Approach

- **Architecture Cloning**
  - Introduced in the latest version of the XL compiler
  - Allows the compiler to target more than one PowerPC processors
  - Additional targets supported : Power4, Power5 and PPC970
  - Generates different instructions optimized for each target
  - Inserts runtime check in program to select the appropriate code path according to the hardware platform
  - To enable architecture cloning, one must compile with –qipa and specify -qipa=clonearch=*“target”* on the link step

Outline

- Motivation
- **Implementation Details**
- Results

How Architecture Cloning Works

- **Architecture Cloning is divided into 2 phases**
  - Analysis phase
  - Transformation phase

Analysis Phase

- **Goal:**
  - Minimize the impact of architecture cloning on link time and executable size by reducing the number of procedures to clone

- **Examines each node in call graph to eliminate candidates**
  - First, it identifies the procedures that cannot be cloned
    - Ex. Procedures not compiled with –qipa, etc.
  - Finally, avoid cloning unprofitable procedures
    - Ex. Procedures marked as having low calling frequency in the call graph, etc.

How To Assist the Analysis

- **Users can instruct the compiler which procedures it should clone or not clone**
  - With compiler suboptions -qipa=cloneproc=*“procname”* and –qipa=nocloneproc=*“procname”*
  - Helpful in cases where 10% of the code is being executed 90% of the time

- **When PDF (Profile-Directed Feedback) is used**
  - the calling frequency is known and thus more accurate
  - More aggressive analysis is performed where it selects from the hottest procedure until a threshold is reached

Transformation Phase

- Inserts a platform detection routine at the program’s entry point
- Performs procedure cloning on the candidates
- Updates the call graph and inserts runtime checks in the program for selecting the right path
- Put the cloned procedures in a separate compilation unit

Insert Platform Detection Routine

- **For the generated binary to determine the platform at runtime**
  - Identify the entry point of the program from the call graph
  - Insert a platform detection routine at the beginning of the entry point
  - This routine obtain processor and OS information from the system
  - The returned result is stored into a global variable to be used for the runtime checks

```
Ex. int main() {
        system_arch = xl_platform_detection()
        ..
        if (system_arch == Pwr4)
            foo@pwr4()
        else
            foo()
        ..
     }
```

Procedure Cloning

- **Why create duplicate procedure copies?**
  - For TPO to apply different architectural-specific optimizations on each copy
  - For TOBEY Backend to generate different instructions and scheduling for each copy

- **The call graph is traversed from top down to find the candidate**
  - remap the parameters and duplicate the body of the procedure
  - add a suffix to the cloned procedure to indicate its target

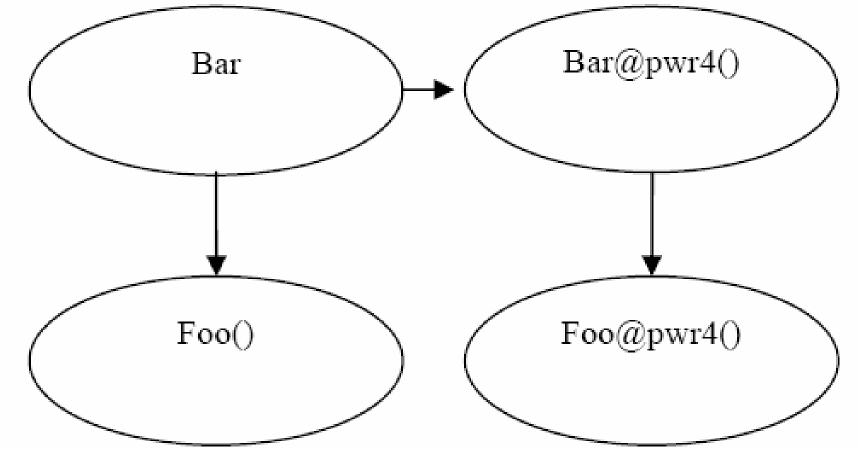
Update Call Graph

- **Attempts to divide the call graph into different sub-graphs**
  - one sub-graph contains the cloned procedures
  - the other sub-graph contains the original procedures

- **In another words, the cloned callers invoke the cloned procedure directly instead of calling the original procedure**

- **The decision for selecting the code path is moved as high as possible in the call graph**

- **Therefore less runtime checks are inserted, and they are unlikely to be placed in the hot procedures**



Final Step of Transformation Phase

- **Put the cloned procedures in a separate compilation unit**
  - TPO applies architectural specific optimizations differently on those cloned procedures

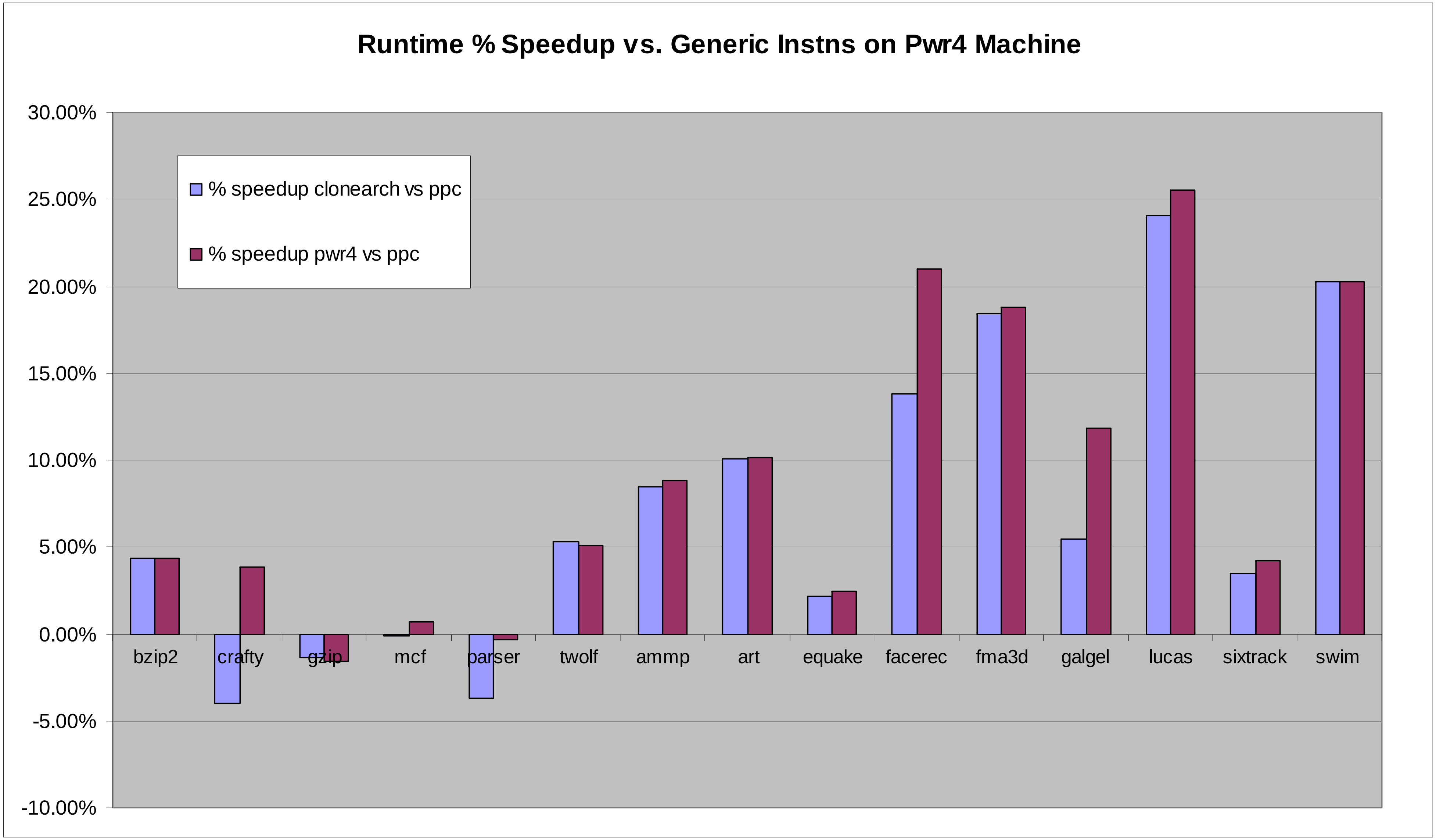
- **TPO sends a separate Wcode with a different architecture setting for this compilation unit to TOBEY**
  - TOBEY generates and schedules the instructions based on the architecture setting from the given Wcode

- **The resulting code is partitioned in memory such that the procedures for each target are contiguous**
  - minimizes the performance impact due to code growth with “demand paging”

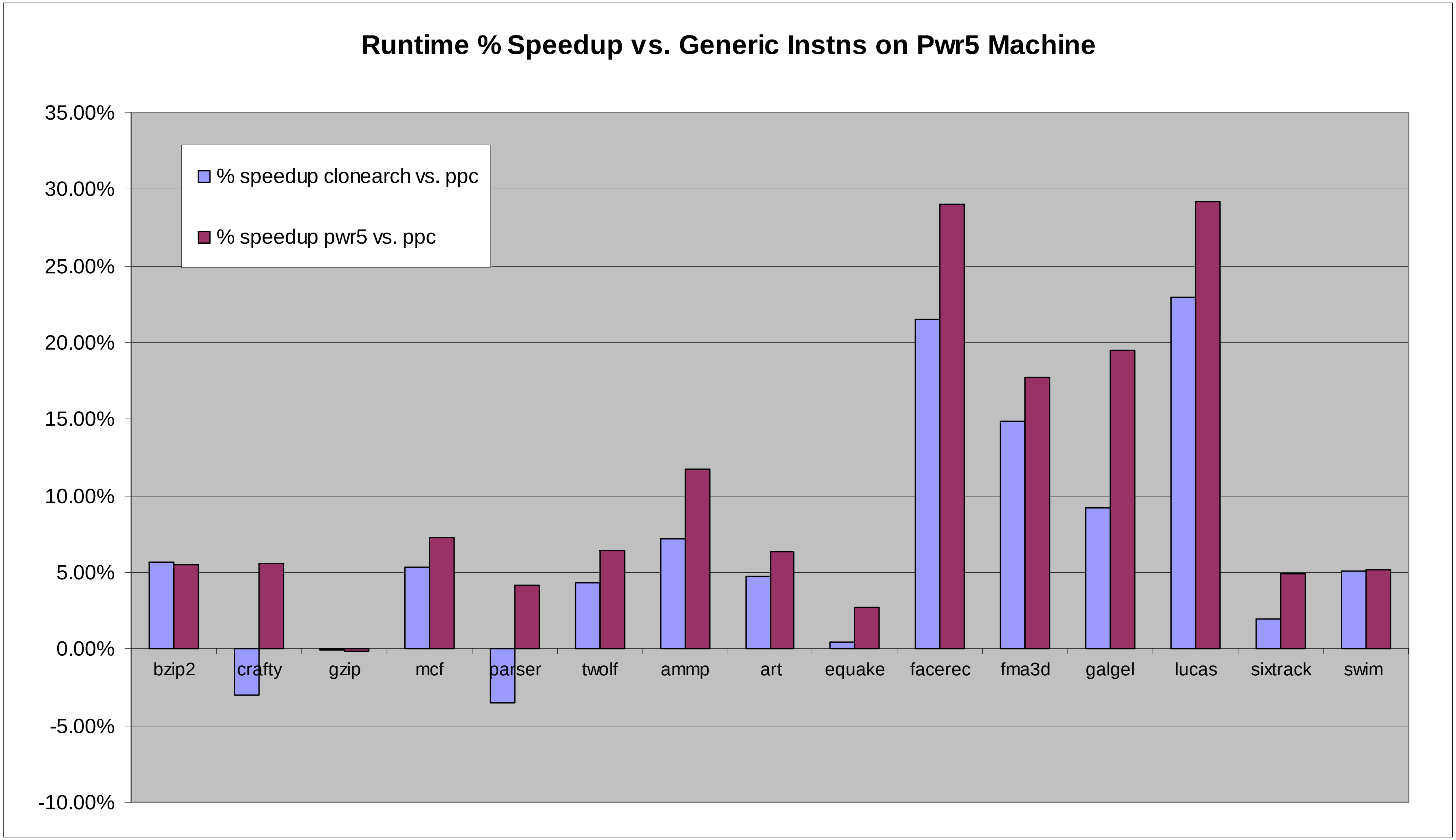
Outline

- Motivation
- Implementation Details
- Results

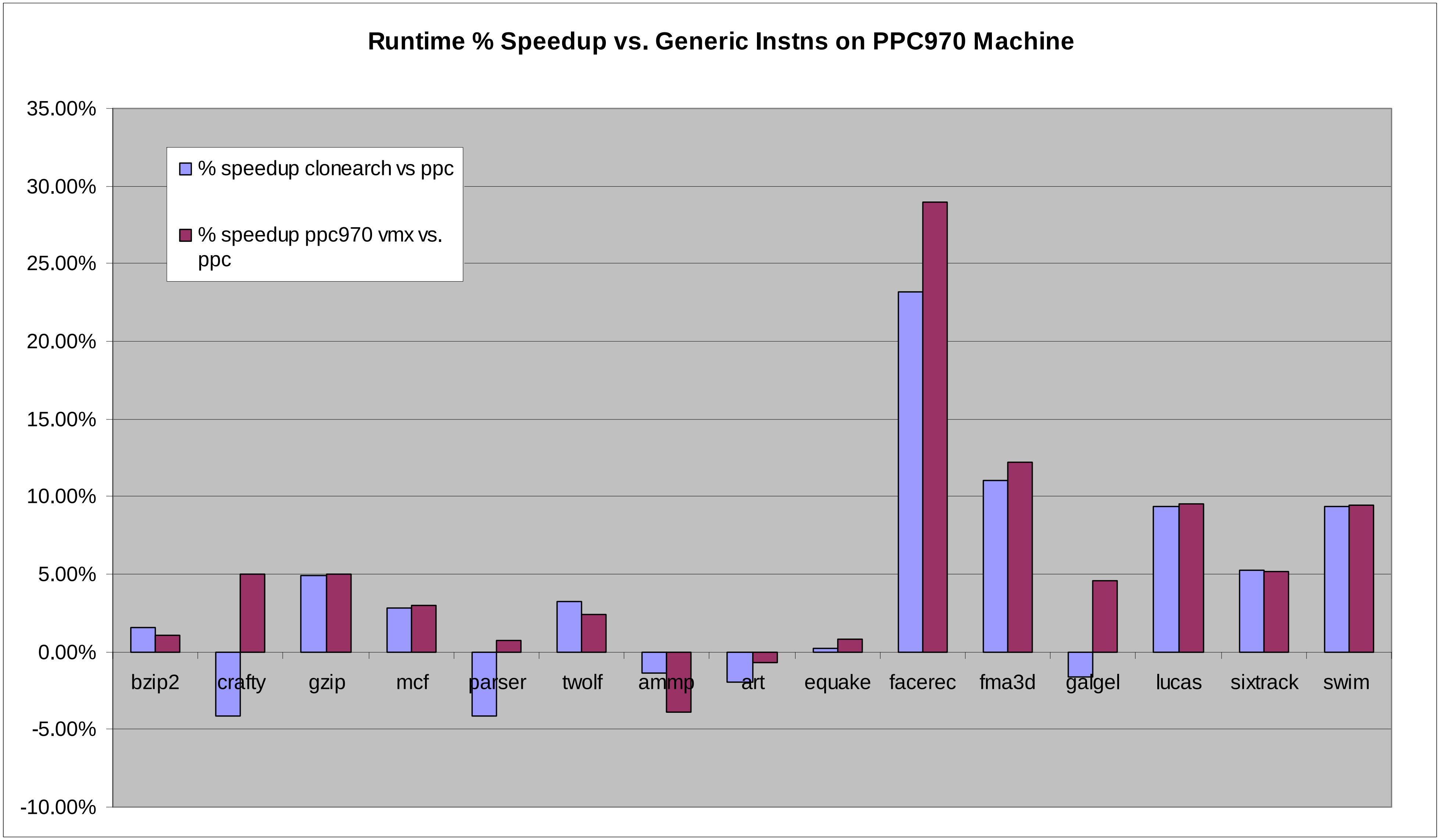
Runtime Comparison : Power4



Runtime Comparison : Power5



Runtime Comparison : PPC970 VMX



Observations

- **Architecture Cloning delivers similar performance compare to the binary optimized for one platform in most benchmarks across all 3 platforms**
  - crafty and parser under investigation

- **Some benchmarks benefit tremendously with architecture-specific instructions and scheduling**
  - Ex. facerec, fma3d, lucas

Conclusions

Architecture Cloning:

- Takes advantage of the latest PowerPC processor features
- Also maintains compatibility with older PowerPC processors
- All within a single code base and single executable

Questions?