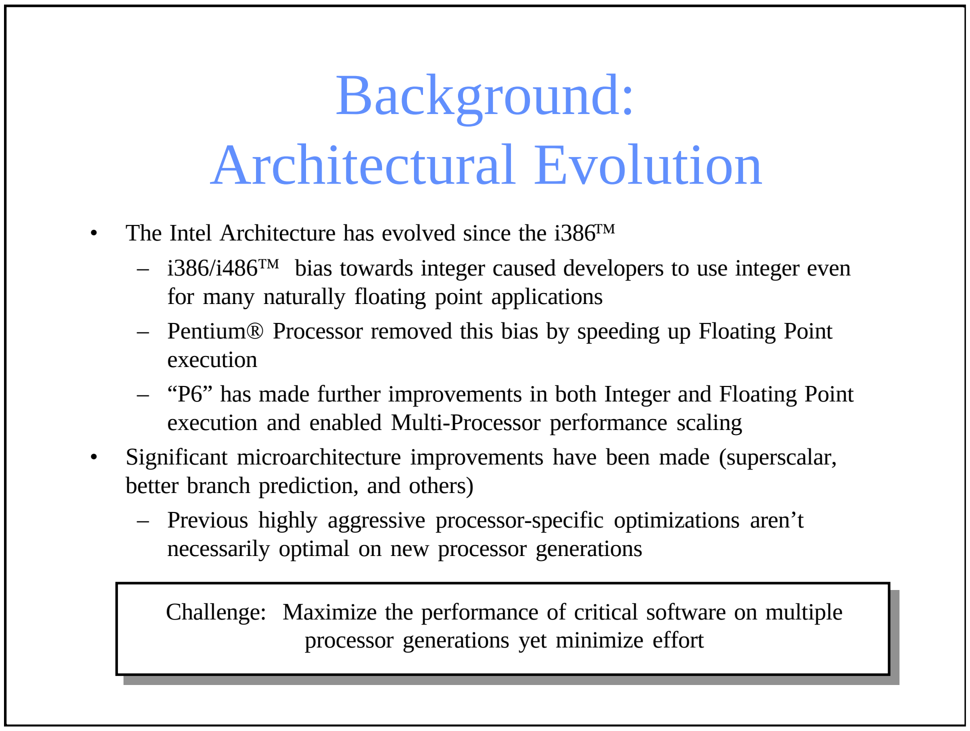
Software Migration and Trade-offs of Processor-specific Tuning



Background: Architectural Evolution

- The Intel Architecture has evolved since the i386™
  - i386/i486™ bias towards integer caused developers to use integer even for many naturally floating point applications
  - Pentium® Processor removed this bias by speeding up Floating Point execution
  - “P6” has made further improvements in both Integer and Floating Point execution and enabled Multi-Processor performance scaling
- Significant microarchitecture improvements have been made (superscalar, better branch prediction, and others)
  - Previous highly aggressive processor-specific optimizations aren’t necessarily optimal on new processor generations

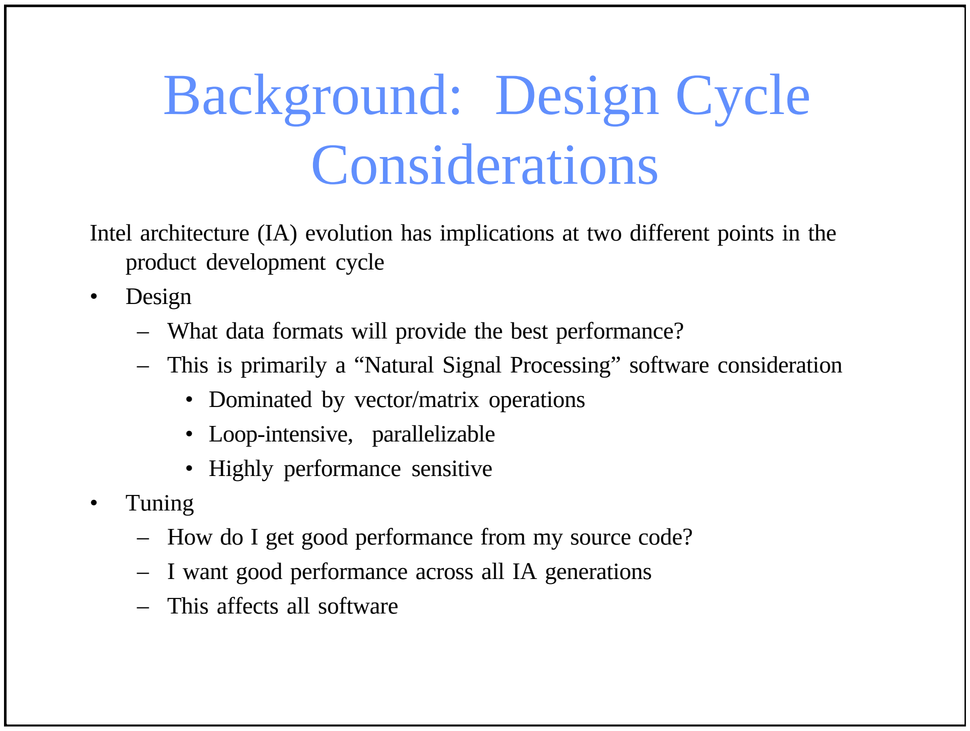
Challenge: Maximize the performance of critical software on multiple processor generations yet minimize effort

The software developed for the Intel Architecture has, naturally, been focused in the area where the architecure performed the highest. That typically has been the integer area. This has resulted in algorithm development that has been forced into a “non-natural” implementation.

The arrival of the Pentium® Processor improved the performance of the Floating Point unit to the point where many applications could move their integer workarounds into floating point where it should have always been.

P6 completes this process by bringing a “high performance” FP and integer unit together.

Older code optimizations may be capable of being improved on the newer Pentium® Processor and P6 architectures by creating a more appropriately targeted or balanced algorithm.

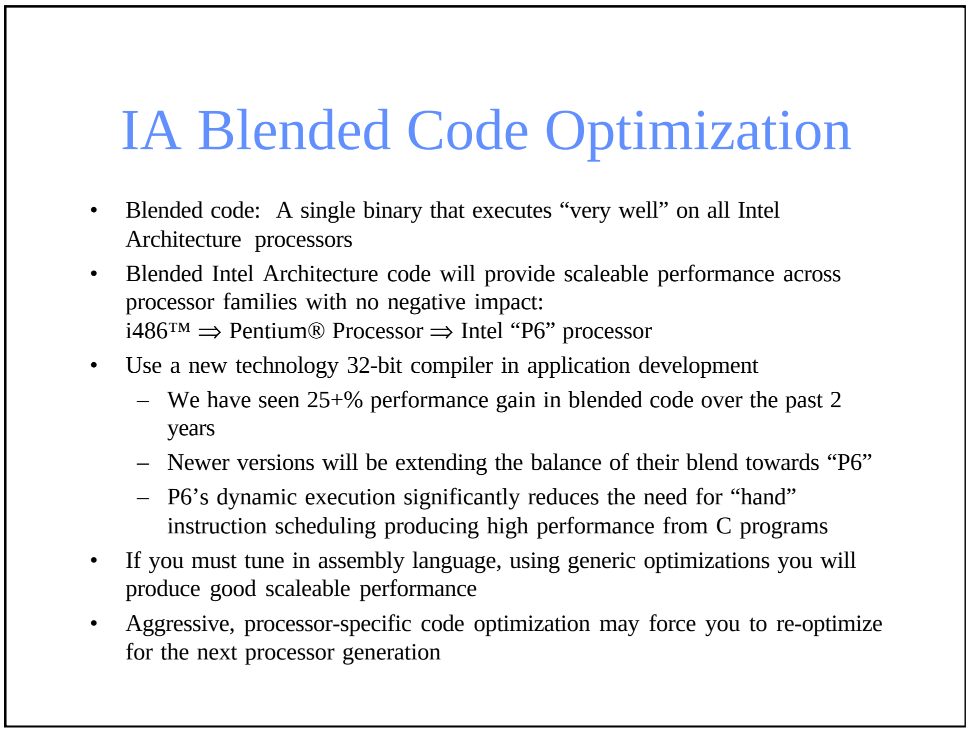


Background: Design Cycle Considerations

Intel architecture (IA) evolution has implications at two different points in the product development cycle

- Design
  - What data formats will provide the best performance?
  - This is primarily a “Natural Signal Processing” software consideration
    - Dominated by vector/matrix operations
    - Loop-intensive, parallelizable
    - Highly performance sensitive
- Tuning
  - How do I get good performance from my source code?
  - I want good performance across all IA generations
  - This affects all software

The most approiate time for application tuning is at design time. Selecting an algorithm and architecture that leads to high performance is easier than trying to retroactively tune an existing code base. However, reality indicates that code tuning is always required and occasionally vital to the success of a coding effort.



IA Blended Code Optimization

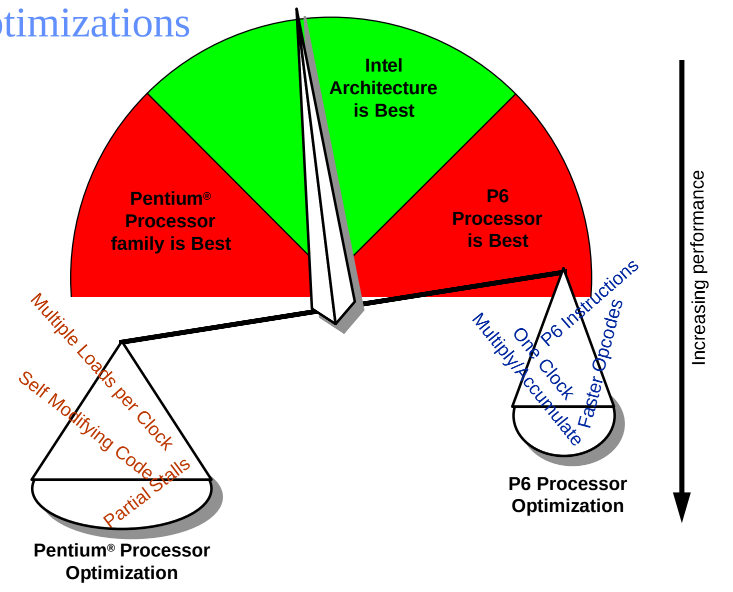
- Blended code: A single binary that executes “very well” on all Intel Architecture processors
- Blended Intel Architecture code will provide scaleable performance across processor families with no negative impact:
  i486™ ⇒ Pentium® Processor ⇒ Intel “P6” processor
- Use a new technology 32-bit compiler in application development
  - We have seen 25+% performance gain in blended code over the past 2 years
  - Newer versions will be extending the balance of their blend towards “P6”
  - P6’s dynamic execution significantly reduces the need for “hand” instruction scheduling producing high performance from C programs
- If you must tune in assembly language, using generic optimizations you will produce good scaleable performance
- Aggressive, processor-specific code optimization may force you to re-optimize for the next processor generation

In developing code for Intel architecture processors, opting to implement algorithms and code sequences that are “blended” will ensure that the code will scale on present and future processors.

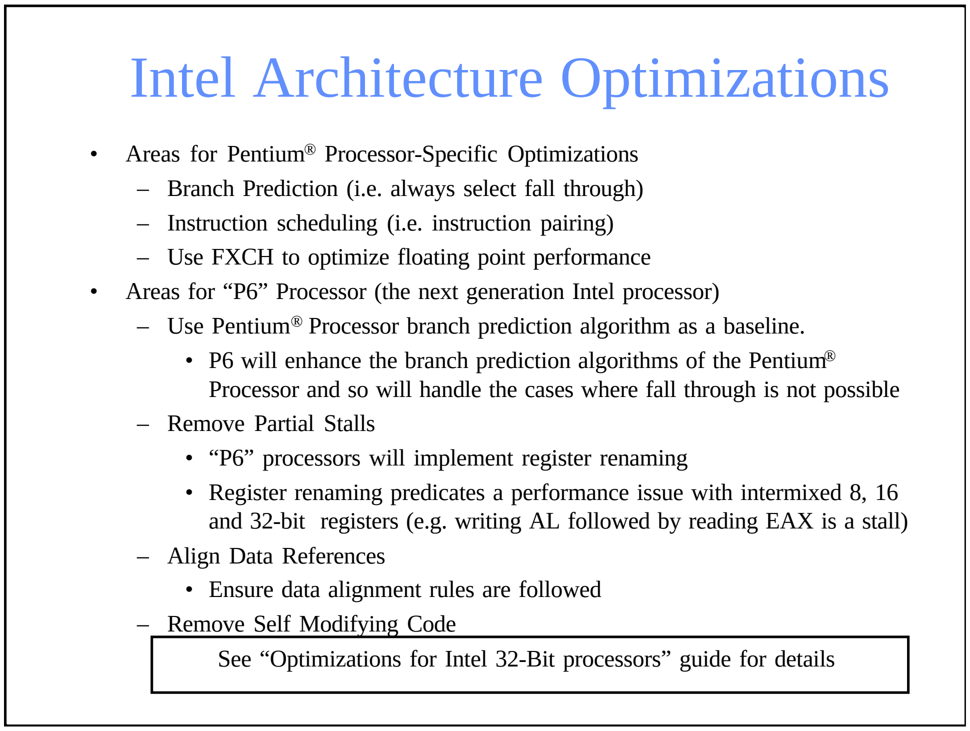
New compilers implement a blended code generation process that has the ability to shift the center of the blend. Today the blend is migrating from an i486™ processor center to a Pentium® Processor center. We will see this center point move towards the P6 over the next 2 years.

Caution! If you need to tune code for a particular processor be it an i486 processor or the Pentium® processor, you may need to recode the tuned sections for a future procesor architecture. The more aggressive the tuning, the higher the certainty that recoding will be required.

Potential Impact of Processor-Specific Optimizations



A graphical representation of code tuning. The more Pentium® Processor specific optimizations used in a code sequence, the more the P6 performance will deteriorate. i486™ processor performance will deteriorate more also. When the scales are balanced, we have a scalable application.



Intel Architecture Optimizations

- Areas for Pentium® Processor-Specific Optimizations
  - Branch Prediction (i.e. always select fall through)
  - Instruction scheduling (i.e. instruction pairing)
  - Use FXCH to optimize floating point performance
- Areas for “P6” Processor (the next generation Intel processor)
  - Use Pentium® Processor branch prediction algorithm as a baseline.
    - P6 will enhance the branch prediction algorithms of the Pentium® Processor and so will handle the cases where fall through is not possible
  - Remove Partial Stalls
    - “P6” processors will implement register renaming
    - Register renaming predicates a performance issue with intermixed 8, 16 and 32-bit registers (e.g. writing AL followed by reading EAX is a stall)
  - Align Data References
    - Ensure data alignment rules are followed
  - Remove Self Modifying Code

See “Optimizations for Intel 32-Bit processors” guide for details

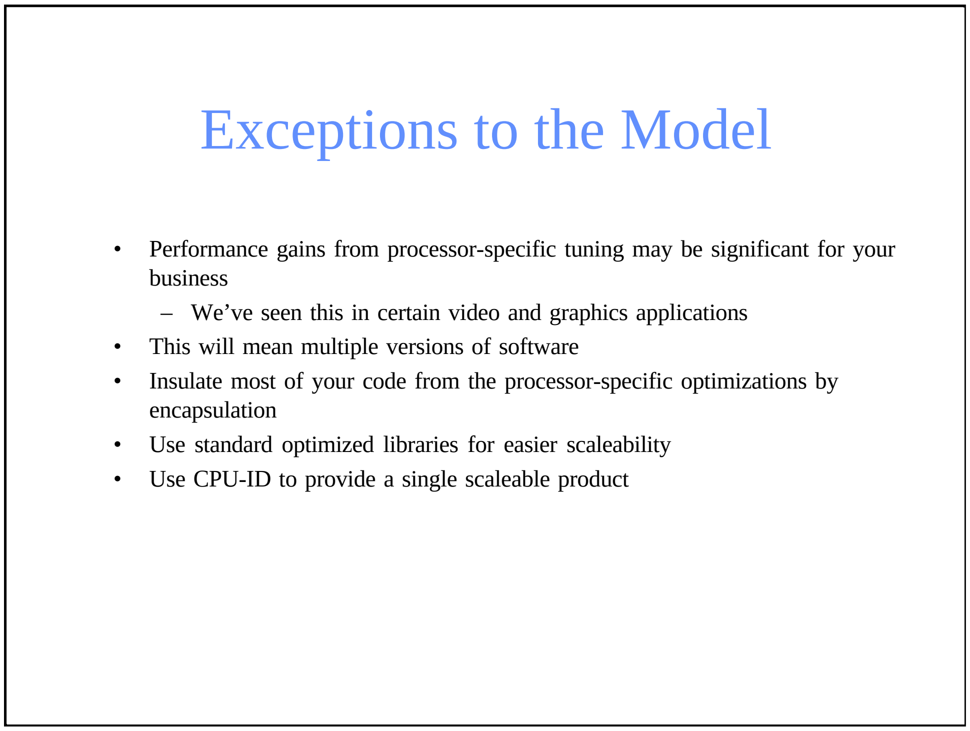
The three main areas for Pentium® Processor code optimizations are: Branch Prediction, Instruction Pairing and Floating Point pipelining.

These optimizations are still good optimizations for the P6. The P6 enhances the Pentium® Processor branch prediction algorithms so branches that cannot be optimized for the Pentium® Processor will be handled better by P6.

P6 performance is reduced when it encounters a Partial Stall. This caused by writing one of the general purpose registers in its 8/16-bit form and then reading it in its 32-bit form. These sequences should be removed from performance-critical sections of code.

Ensure data access occur on the appropriate alignment.

Remove self modifying code. This causes the P6 pipeline to be flushed when detected.



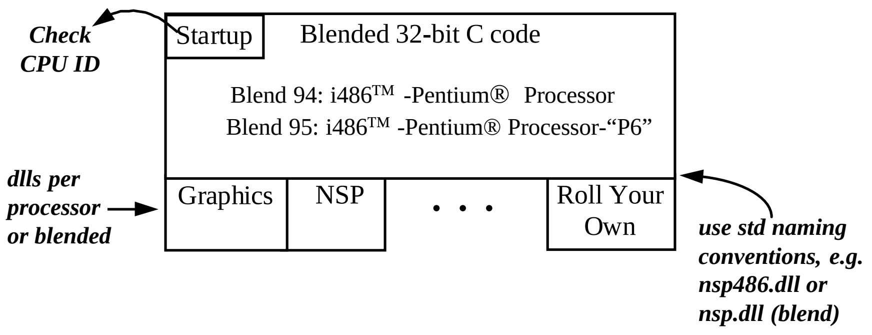
Exceptions to the Model

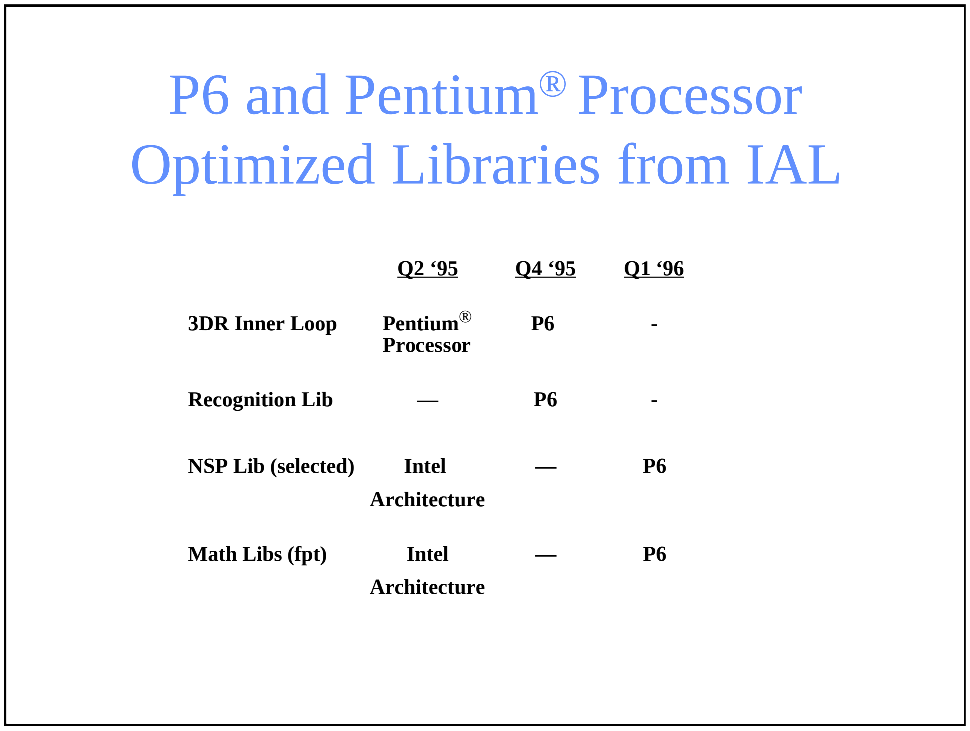
- Performance gains from processor-specific tuning may be significant for your business
  - We’ve seen this in certain video and graphics applications
- This will mean multiple versions of software
- Insulate most of your code from the processor-specific optimizations by encapsulation
- Use standard optimized libraries for easier scaleability
- Use CPU-ID to provide a single scaleable product

The level to which code is tuned is determined by the business needs of a particular company. It may be needed to make an application even possible or acceptable. The implications of this are multiple versions of software and multiple coding/tuning efforts.

It is best to ensure that the heavily tuned sections of code are encapsulated in “removable” modules or libraries. Modern processors support the CPUID instruction and provide specific processor feature identification. Combining this processor feature with the current models for operating systems that allow loadable modules and libraries will ensure that optimum performance is maintained accross a mixture of application and processor bases. This will also reduce software maintainace and support costs.

Insulating code: Sample Encapsulation Model


Check CPU ID
Startup
Blended 32-bit C code
Blend 94: i486™ -Pentium® Processor
Blend 95: i486™ -Pentium® Processor-"P6"
dlls per processor or blended
Graphics
NSP
. . .
Roll Your Own
use std naming conventions, e.g. nsp486.dll or nsp.dll (blend)



P6 and Pentium® Processor Optimized Libraries from IAL

	Q2 '95	Q4 '95	Q1 '96
3DR Inner Loop	**Pentium® Processor**	**P6**	-
Recognition Lib	—	**P6**	-
NSP Lib (selected)	**Intel Architecture**	—	**P6**
Math Libs (fpt)	**Intel Architecture**	—	**P6**

Intel has been following the encapsulation and tuning models outlined previously and has developed libraries for specific functions. An example from the table above is 3DR, the three dimensional grahics library. Today this library is being tuned for the Pentium® Processor. Later this year it will be tuned for the P6 processor.

Summary

- Using blended code when compiling or tuning will yield a binary that executes well on all Intel Architecture processors
  - If processor-specific tuning is needed for your business
    - You will need multiple versions of your code
    - Encapsulate if possible to insulate the effect
    - Use standard libraries if possible to facilitate scaleability
    - Use CPU-ID to provide a single scaleable product