Software Performance Engineering - SPE

Optimized C++

Ed Keenan

6 March 2019

13.0.6.5.6 Mayan Long Count

DePaul University

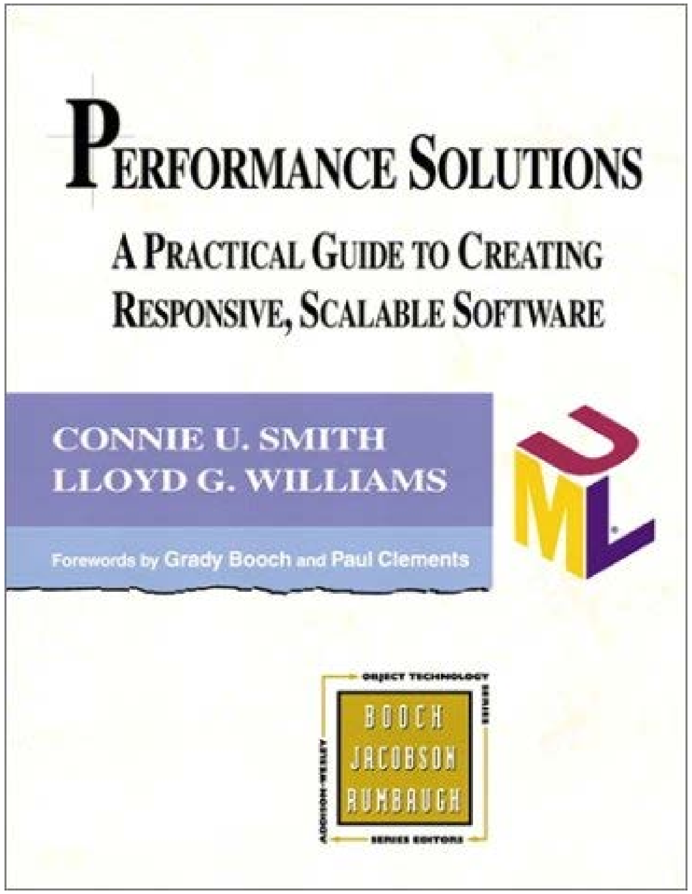
Goals

- What is SPE?
- Performance
  - Models
  - Responsiveness
  - Scalability
- Myths
- Strategies
- Modeling
- Distributed Issues

How do you confuse a optimization class?

Reference material

- Book:
  - Performance Solutions: A practical guide to creating responsive, scalable software
- Articles:
  - Five Steps to Solving Software Performance Problems
  - Making the Business Case for Software Performance Engineering
  - Best Practices for Software Performance Engineering
- Authors:
  - Connie Smith and Lloyd Williams


PERFORMANCE SOLUTIONS

A PRACTICAL GUIDE TO CREATING RESPONSIVE, SCALABLE SOFTWARE

CONNIE U. SMITH
LLOYD G. WILLIAMS

Forewords by Grady Booch and Paul Clements

Software Performance Engineering

- Software performance engineering (SPE)
  - A systematic, quantitative approach to constructing software systems that meet performance objectives.
- SPE prescribes
  - principles for creating responsive software
  - the data required for evaluation
  - procedures for obtaining performance specifications
  - guidelines for conducting performance evaluation at each development stage

DePaul University

Model Based

- SPE is model based
  - Building and analyzing models of the proposed software
  - Explore model’s characteristics
    - Determine whether it meets the requirements before we build the system
- Techniques are neither new or revolutionary
  - Evolved from proven quantitative disciplines

Performance

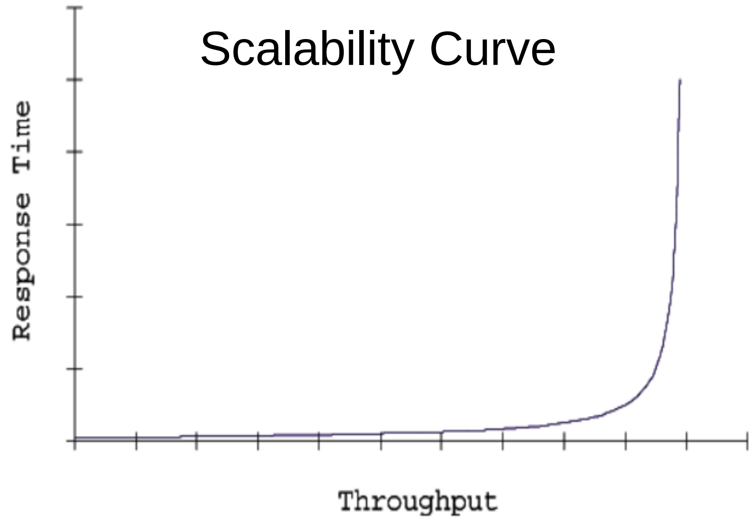
- Performance is the degree to which a software system or component meets its objectives for timelines
- Performance is an indicator of how well a software system meets its requirement on timeliness
  - Response time
  - Throughput time

Responsiveness

- Responsiveness is the ability of a system to meet its objectives for response time or throughput.
- Responsiveness has both an objective and a subjective component
- Example:
  - improve the perceived responsive of a web application by present the user-writable fields first.

Scalability

- Scalability is the ability of a system to continue to meet its response time and throughput objectives as the demand for the software functions increases.
- At some point a small increase in load begins to have a great effect on response time.



DEPAUL UNIVERSITY

Causes of Performance Loss

- Problems introduced at beginning of project
  - Attitude:
    - Let's get it done
    - If performance problem – fix it later
- Many encourage this attitude
  - Get the structure and flow working
  - We'll focus on optimizing the small part that's problematic
    - We'll be more efficient

Performance Myth

- Myth 1:
  - Based on the assumption that you need something to measure before you can begin to manage performance
    - *It is not possible to do anything about performance until you have something executing to measure*
- Reality
  - You don’t need to wait
  - Use performance models at the early architectural and early design phases
  - Models allow you estimate the performance and predict the final performance

Performance Myth

- Myth 2:
  - Based on the fear that adding performance management to the software process will delay project completion
    - *Managing performance takes too much time*
- Reality
  - Do not automatically require addition time
  - Amount of effort is required is proportional to the level of performance risk your system
  - Fixing and identifying problems at the early stage saves time in the end

Performance Myth

- Myth 3:
  - Based on the fear that adding performance modeling will take too much time and consume too many project resources
    - *Performance models are complex and expensive to construct*
- Reality
  - Simple models work extremely well
  - Simple models are inexpensive to construct and evaluate

DEPAUL UNIVERSITY

Getting it Right

- Even if you Rely on hardware to solve performance problems
  - Use performance models early
    - Before other options are closed
  - Verify that this a cost-effective solution
    - IBM server example
- Sometimes – End to End performance testing is not possible
  - Performance models are the only option

Proactive performance

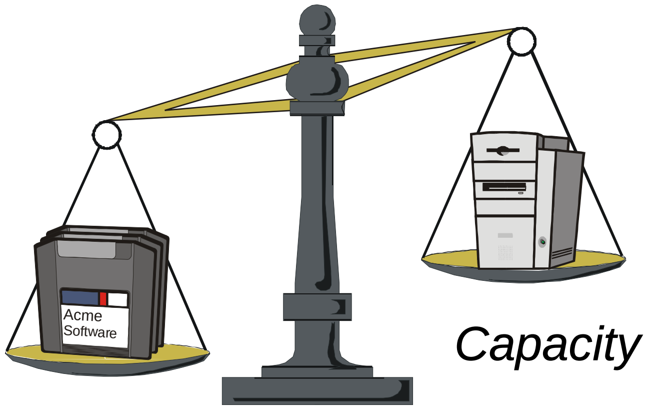
- The use of new software technology requires careful attention to performance until the performance aspects of the new technology are understood
- Intuition about performance problems is not sufficient
  - Quantitative assessments are necessary to assess performance risks
- Performance engineering reactively on systems that use unfamiliar technology
  - Probability of performance failure is much higher

SPE

- Answers the following questions:
  - Will your users be able to complete tasks in the allotted time?
  - Are your hardware and network capable of supporting the load?
  - What response time is expected for key tasks?
  - Will the system scale up to meet your future needs?

Performance Balance

- Detect problems early
- Quantitative Assessment
  - Begins early, frequency matches system criticality
- Often find architecture and design alternatives with lower resource requirements
- Select cost-effective performance solutions early



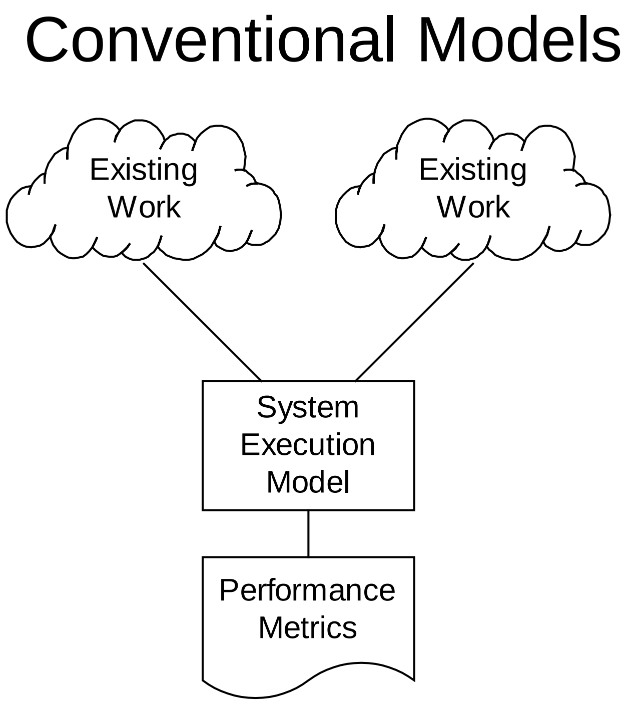
Capacity

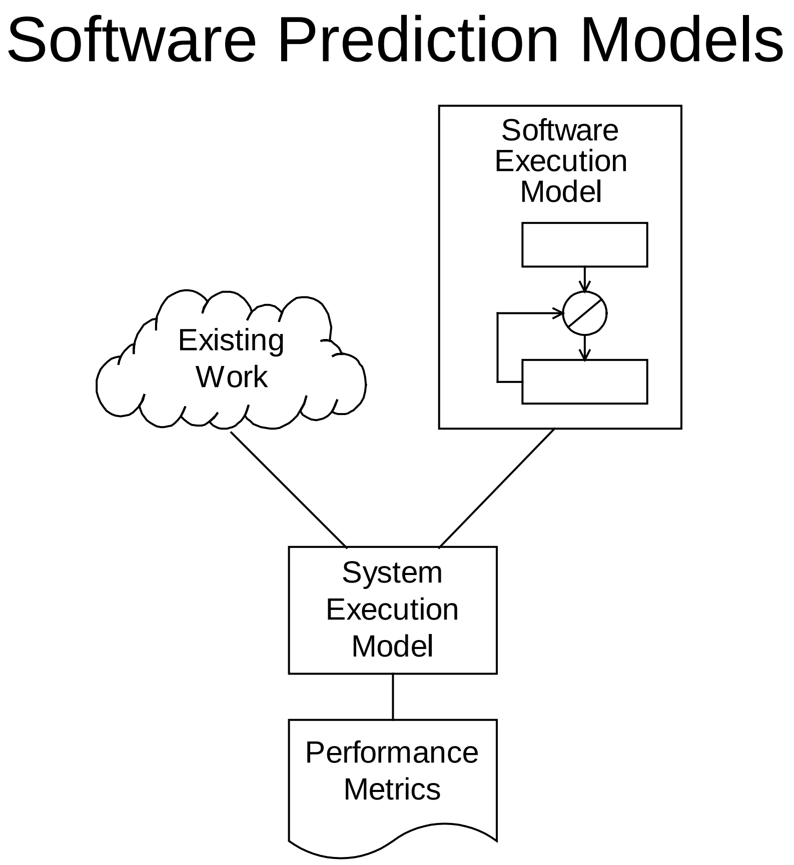
Resource Requirements

SPE Modeling Strategies

- Simple-Model Strategy
  - Simplest possible model that identifies problems with the system architecture, design or implementation plans
- Best- and Worst-Case Strategy
  - Best- and worst-case estimates of resource requirements to establish bounds on expected performance and manage uncertainty in estimates
- Adapt-to-Precision Strategy
  - Match the details represented in the models to your knowledge of the software processing details

SPE Model-based Approach





DePaul University

SPE Model Requirements

- Low overhead
  - use the simplest possible model that identifies problems
- Accommodate:
  - incomplete definitions
  - imprecise performance specifications
  - changes and evolution
- Goals:
  - initially distinguish between "good" and "bad"
    - later, increase precision of predictions
  - provide decision support

SPE Process Steps

1. Assess performance risk
   - Identify critical use cases
2. Select key performance scenarios
   - Establish performance objectives
3. Construct performance models
   - Determine software resource requirements
4. Evaluate the models
   - Verify and validate the models

Workload Data

- Pareto principle (80-20 rule)
  - Typically 80% of the software execution will run in 20% of the code
- First: scenarios of *typical* activity
  - Number of concurrent users
  - Request arrival rates
  - Performance goals
- Later:
  - add large scenarios, critical scenarios, etc.

DEPAUL UNIVERSITY

Software Specifications

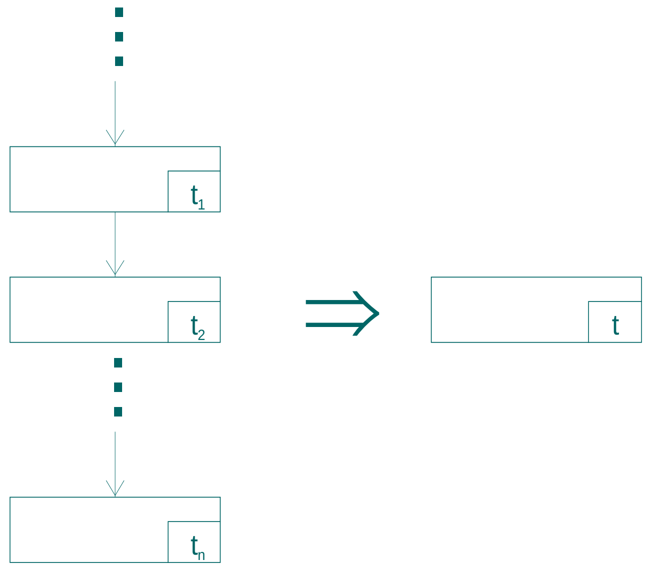
- Execution paths for scenarios of interest
- Objects / methods to be executed
  - probability of execution
  - number of repetitions
  - protocol
- Slow Dependencies
  - Networking
  - Database accesses
- *Level of detail increases as development progresses*

Understand Resource Usage

- CPU
  - Work Units
  - Number of instructions executed or measurements of similar software
- I/O
  - External devices
- Communication
  - Networking
  - Database calls

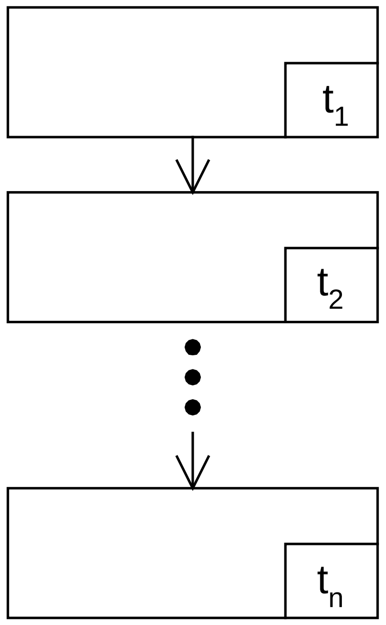
Software Model Solutions

- Types of solutions:
  - Best case - shortest path in graph
  - Worst case - longest path in graph
  - Average
  - Variance
- Approach
  - Repeat reduction rules on typical structures until graph contains one node with the computed solution
  - Apply reductions to each resource specification (t), then combine the results



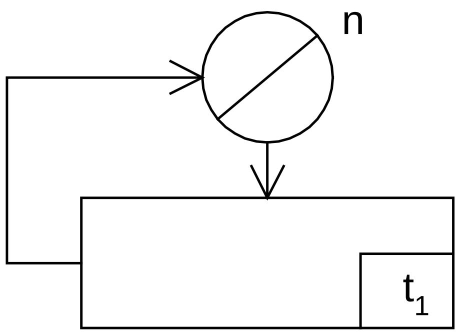
DePaul University

Simple Reduction Rules:
Average Analysis



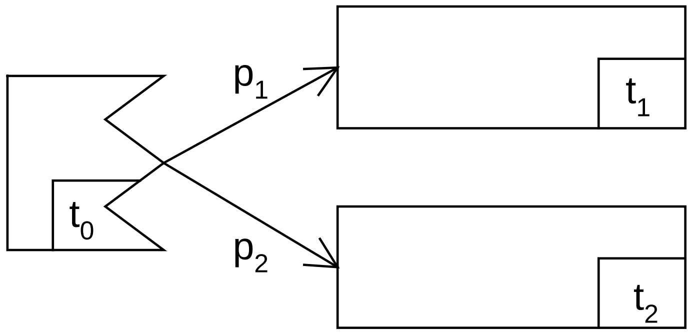
$t = t_1 + t_2 + \ldots + t_n$

Sequential Structures



$t = nt_1$

Loop Structures



$t = t_0 + p_1t_1 + p_2t_2$

Case Nodes

DePaul University

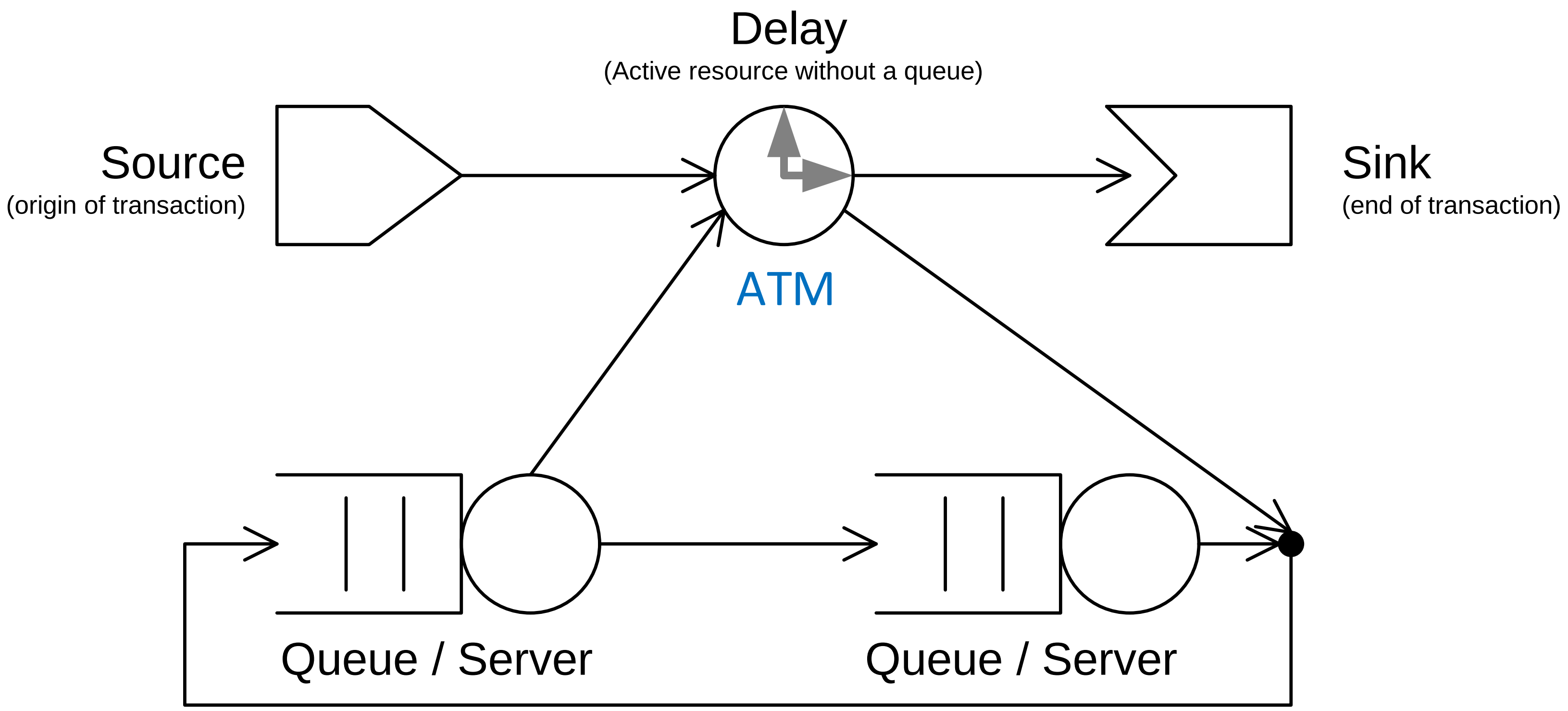
System Execution Model

- Characterizes performance in the presence of factors that could cause contention for resources
  - Multiple workloads
  - Multiple users

System Execution Model

- Provides additional information
  - Metrics that account for resource contention
  - Sensitivity of performance metrics
    - to variations in workload composition
  - Scalability of the hardware and software
    - Identification of bottleneck resources
  - Comparative data on options:
    - workload changes, software changes, hardware upgrades, and various combinations of each

ATM – example:
System Performance Model



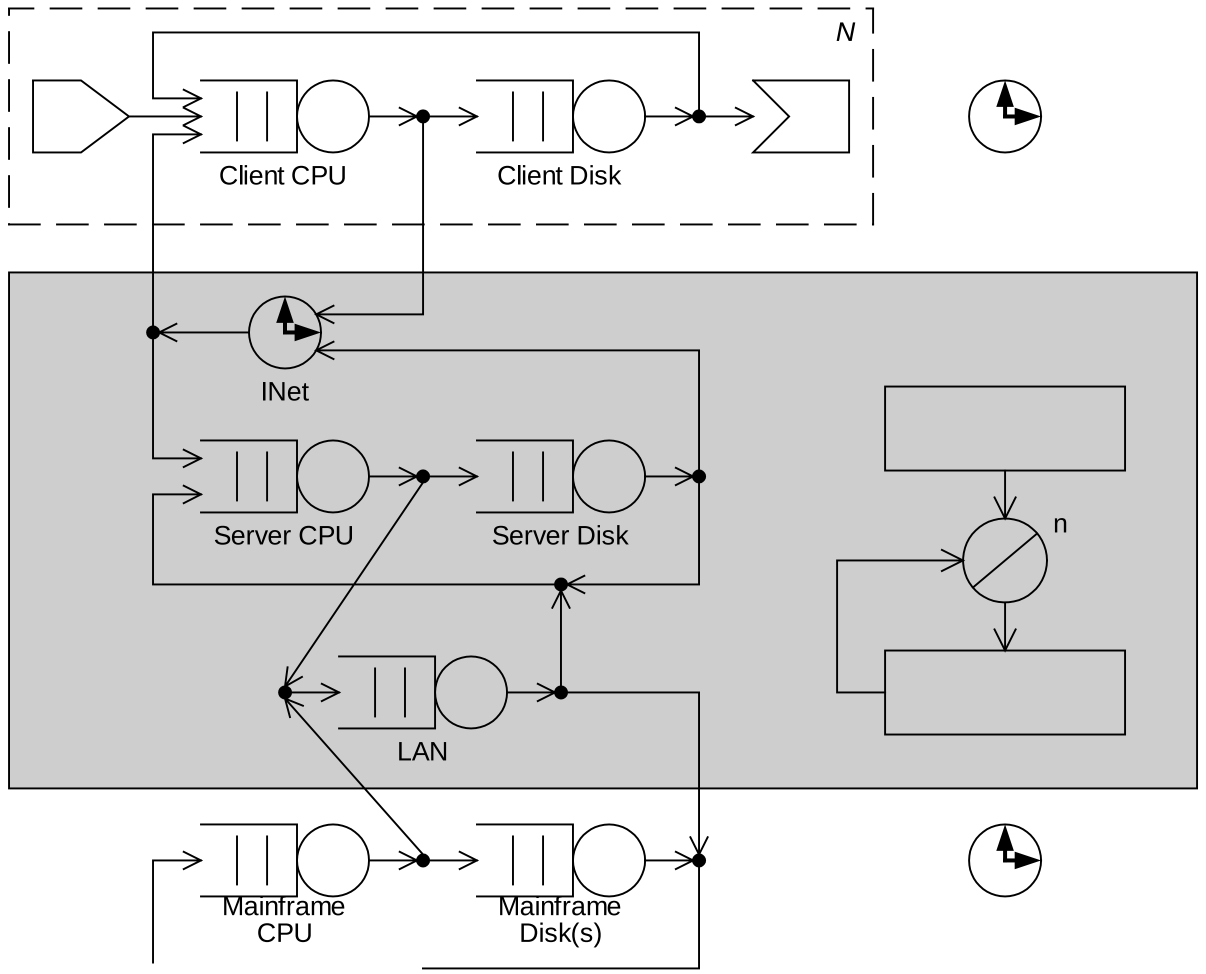
System Performance Model

- What is the System Performance model?
  - You are making a simulator
    - Simulate the load with the models
    - Can be as accurate or inaccurate as you desire
    - Look for trends

Distributed Modeling?

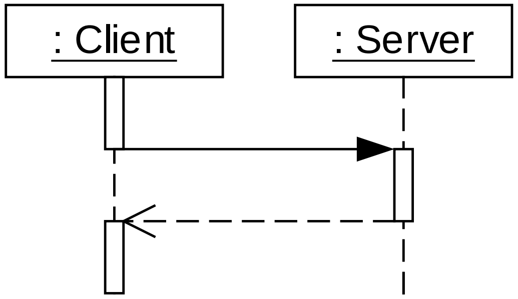
- Simple-Model strategy
  - Start with software execution model
  - One scenario/processor at a time
- Approximate delays
  - Communication/ synchronization
- If software execution model shows no problems, proceed to
  - System execution model
  - Advanced system execution model

Partitioning the Model

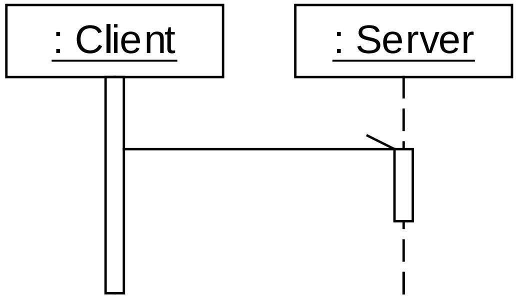


Estimate the Delay for System Interactions

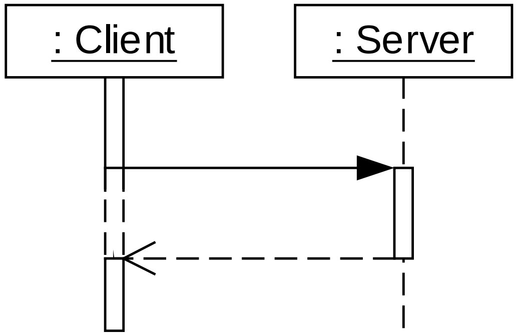
Message Types



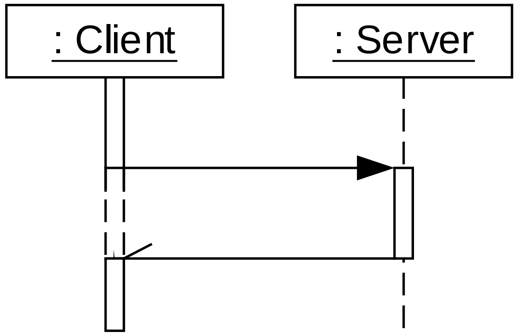
Synchronous



Asynchronous



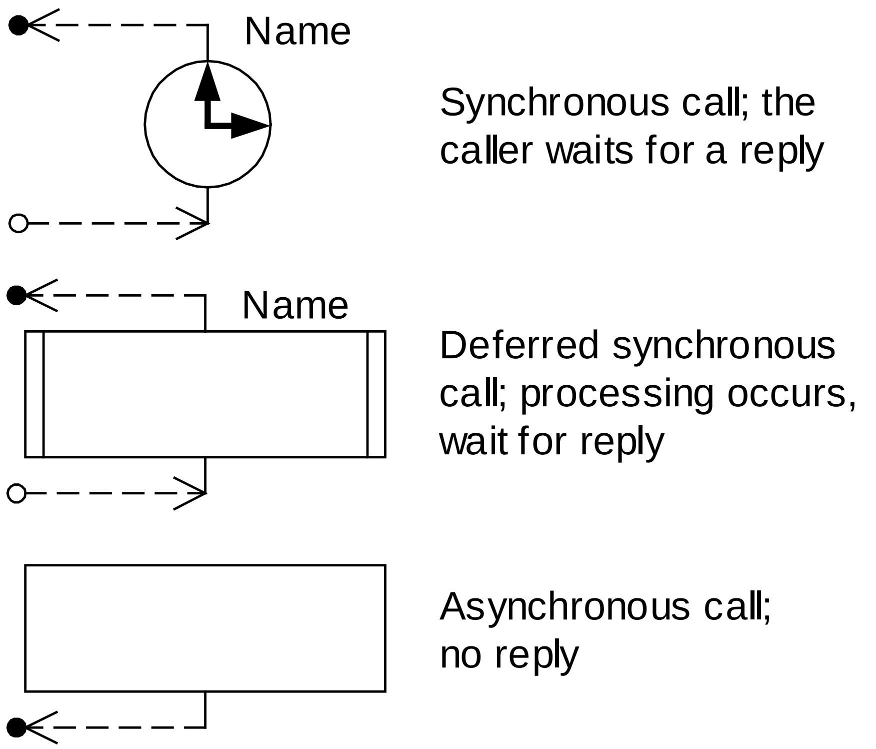
Deferred Synchronous

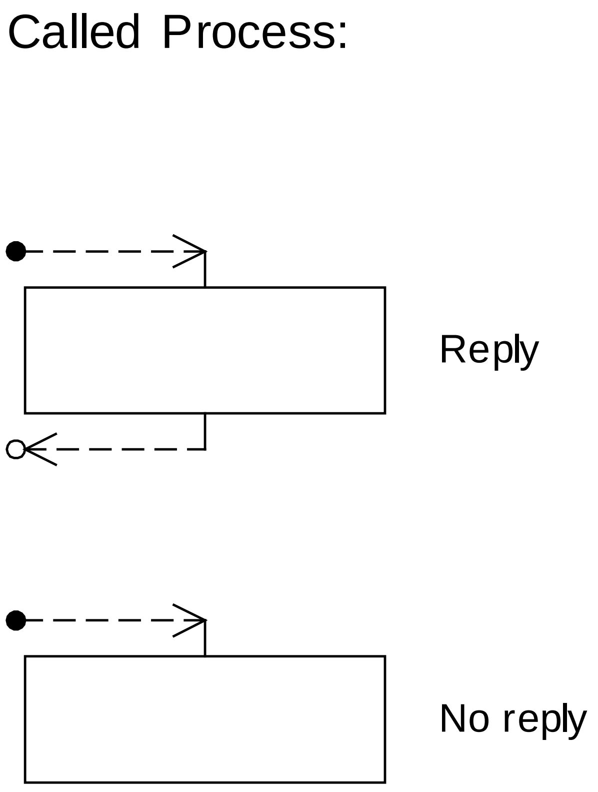


Asynchronous Callback

DePaul University

Synchronization Nodes


Calling Process:
Name
Synchronous call; the caller waits for a reply
Name
Deferred synchronous call; processing occurs, wait for reply
Asynchronous call; no reply



DEPAUL UNIVERSITY

Summary

- Early modeling is important for performance related design
- Distributed systems
  - Use Simple-Model strategy
- With SPE you can gain insights into:
  - Performance principles
  - Patterns
  - Anti-patterns
  - That affect your design

Thank You!

- Questions?

DePaul University

Profiler

Optimized C++

Ed Keenan

DEPAUL UNIVERSITY

Goals

- Basics
  - Profiling
  - Hot Spot
  - Instrumenting
  - Bottlenecks
- VTune Example
  - Sampling
  - Call Graph
  - Event Based
- Popular Profilers
  - Intel
  - Microsoft
  - Apple

To profile or not to profile?

Review Material

- General References from Web
- Basics of VTune™ Performance Analyzer
  - Intel Software College
- Profiling tools
  - By Vitaly Kroivets
- Great Lecture on internals
  - Software Performance Profiling
    - MAE Presentation - Nagy Mostafa

Profiler Basics

Efficient Code?

- Efficiency
  - Describe properties of an algorithm relating to how much of various types of resources it consumes
- Algorithmic efficiency
  - Thought of as analogous to engineering productivity for a repeating or continuous process.
  - Goal is to reduce resource consumption
    - Time to completion to some optimal level

Profiling

- Software profiling
  - Form of dynamic program analysis that measures:
    - usage of memory, usage of particular instructions, or frequency and duration of function calls
    - Common use of profiling information is to aid program optimization
  - Profiling is achieved by ***instrumenting*** either the program source code or its binary executable form using a tool called a profiler
- Methodology of the profiler
  - Event-based
  - Statistical
  - Instrumentation
  - Simulation

Gathering Program events

- Profilers use a wide variety of techniques to collect data
  - hardware interrupts
  - code instrumentation
  - instruction set simulation
  - operating system hooks
  - performance counters

Hot Spot Analysis

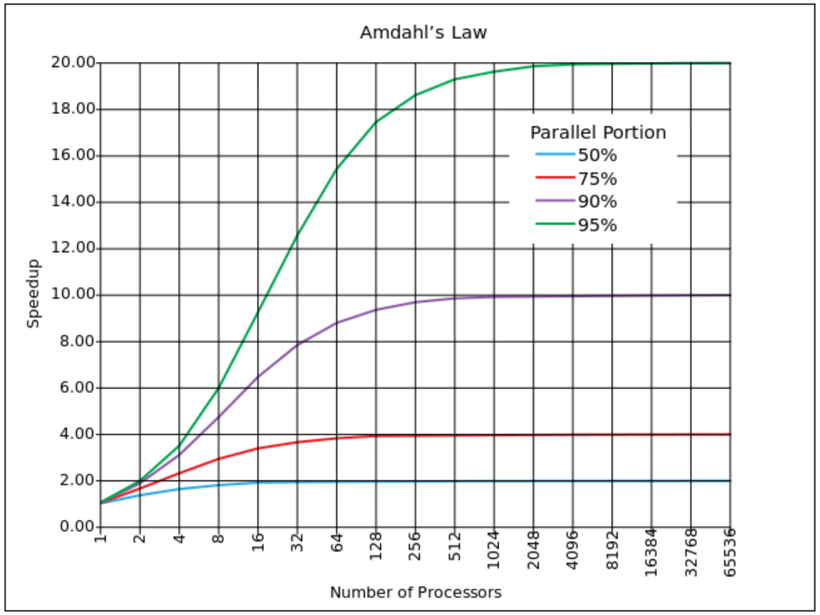
- To diagnose ***hot spots***
  - During actual execution of computer programs
  - Use real or test data
  - Performance analysis under generally repeatable conditions
- Identify ***pinpoint*** sections of the code
  - Specifically targeted programmer optimization without necessarily spending time optimizing the rest of the code
- Re-Running the program
  - Measure of relative improvement can then be determined
  - Determine if the optimization was successful

Amdahl’s Law

- ***Amdahl's law*** also known as ***Amdahl's argument***
  - Named after computer architect Gene Amdahl
  - Finds the maximum expected improvement to an overall system when only part of the system is improved
  - Is often used in parallel computing to predict the theoretical maximum speedup using multiple processors
- Speedup of a program using multiple processors in parallel computing is limited by the time needed for the ***sequential*** fraction of the program
  - Applies to any optimization, single CPU or parallel CPUs

Amdahl's Law Example

- if a program needs 20 hours with a single processor
  - a particular portion of 1 hour cannot be parallelized
  - while the remaining promising portion of 19 hours (95%) can be parallelized
  - How we parallelized execution of this program
  - -> minimum execution time cannot be less than that critical 1 hour
    - Hence the speedup is limited up to 20×



Profiles & Traces confusion

- ***Profile***
  - A statistical summary of the events observed
  - Summary profile information is often annotated against the source code statements where the events occur,
  - Size of measurement data is linear to the code size of the program
- ***Trace***
  - A stream of recorded events
  - Performance problems in parallel programs (waiting for messages or synchronization issues) often depend on the time relationship of events
    - Requiring a full trace to get an understanding of what is happening.

Methods of data gathering

- Sampling Profilers
  - Statistical Profilers
    - Time Based
  - Event Based Profilers
- Instrumenting profilers

Statistical profilers

- A sampling profiler probes the target program's program counter at ***regular intervals*** using operating system *interrupts*
  - Sampling profiles are typically less numerically accurate and specific
  - Allow the target program to run at ***near full speed***
  - Resulting data are not exact, but a statistical approximation
    - Can often provide a more accurate view
- ***Not as intrusive*** to the target program
  - Don't have as many side effects
    - Don't affect the execution speed as much, they can detect issues that would otherwise be hidden.
- Some processors have JTAG interface which samples the program counter in a truly undetectable manner.

Instrumenting profilers

- Some profilers instrument the target program with ***additional instructions*** to collect the required information.
- Instrumenting the program can cause changes in the performance of the program
  - Potentially causing inaccurate results and heisenbugs
    - A ***heisenbug*** (named after the Heisenberg uncertainty principle) is a computer bug that disappears or alters its characteristics when an attempt is made to study it.
- Instrumenting will always have some impact on the program execution, typically always ***slowing*** it.
  - Instrumentation can be carefully controlled to have a minimal impact.
  - Impact on a particular program depends on the placement of instrumentation points and the mechanism used to capture the trace
  - *Our class Timer is an instrument*
- New hardware support for trace capture means that on some targets, instrumentation can be on just one machine instruction.

Performance analysis

- Performance analysis
  - Commonly known as ***profiling***
  - Is the investigation of a program's behavior using information gathered as the program executes.
  - Its goal is to determine which sections of a program to optimize.
- A ***profiler*** is a performance analysis tool
  - Measures the behavior of a program as it executes
    - Particularly the frequency and duration of function calls.
- Performance analysis tools existed at least from the early 1970s.
  - Profilers may be classified according to their output types, or their methods for data gathering.

Self-tuning

- A ***self-tuning*** system is capable of optimizing its own internal running parameters in order to maximize or minimize the fulfillment of an objective function typically the maximization of efficiency or error minimization.
  - Self-tuning systems typically exhibit non-linear adaptive control.
  - Self-tuning systems have been a hallmark of the aerospace industry for decades, as this sort of feedback is necessary to generate optimal multi-variable control for nonlinear processes.
- Game development is now moving to self-tuning schemes.

Bottlenecks

- ***Bottleneck*** is the part of a system which is at capacity.
  - Other parts of the system will be idle, waiting for it to perform its task.
- The process of finding and removing bottlenecks
  - Need to prove their existence
    - By sampling, before acting to remove them.
  - **Avoid** a strong temptation to *guess*.
    - Guesses are often wrong, and investing only in guesses can itself be a bottleneck

Performance Tuning

- Performance tuning is the improvement of system performance
  - Most systems will respond to **increased load** with some degree of decreasing performance.
  - A system's ability to accept higher load is called ***scalability***
  - Modifying a system to handle a higher load is synonymous to performance tuning.
- Problems may be identified by slow or unresponsive systems.
  - Due to high system loading, causing some part of the system to reach a limit in its ability to respond.

Performance Tuning:
Steps

- Generalized steps
  - Assess problem, establish metrics that categorize acceptable behavior.
  - Measure the performance of the system before modification.
  - Identify the bottleneck, is critical for improving the performance.
  - Modify/Refactor that part of the system to remove the bottleneck.
  - Measure the performance of the system after modification.

Profiler Ouputs

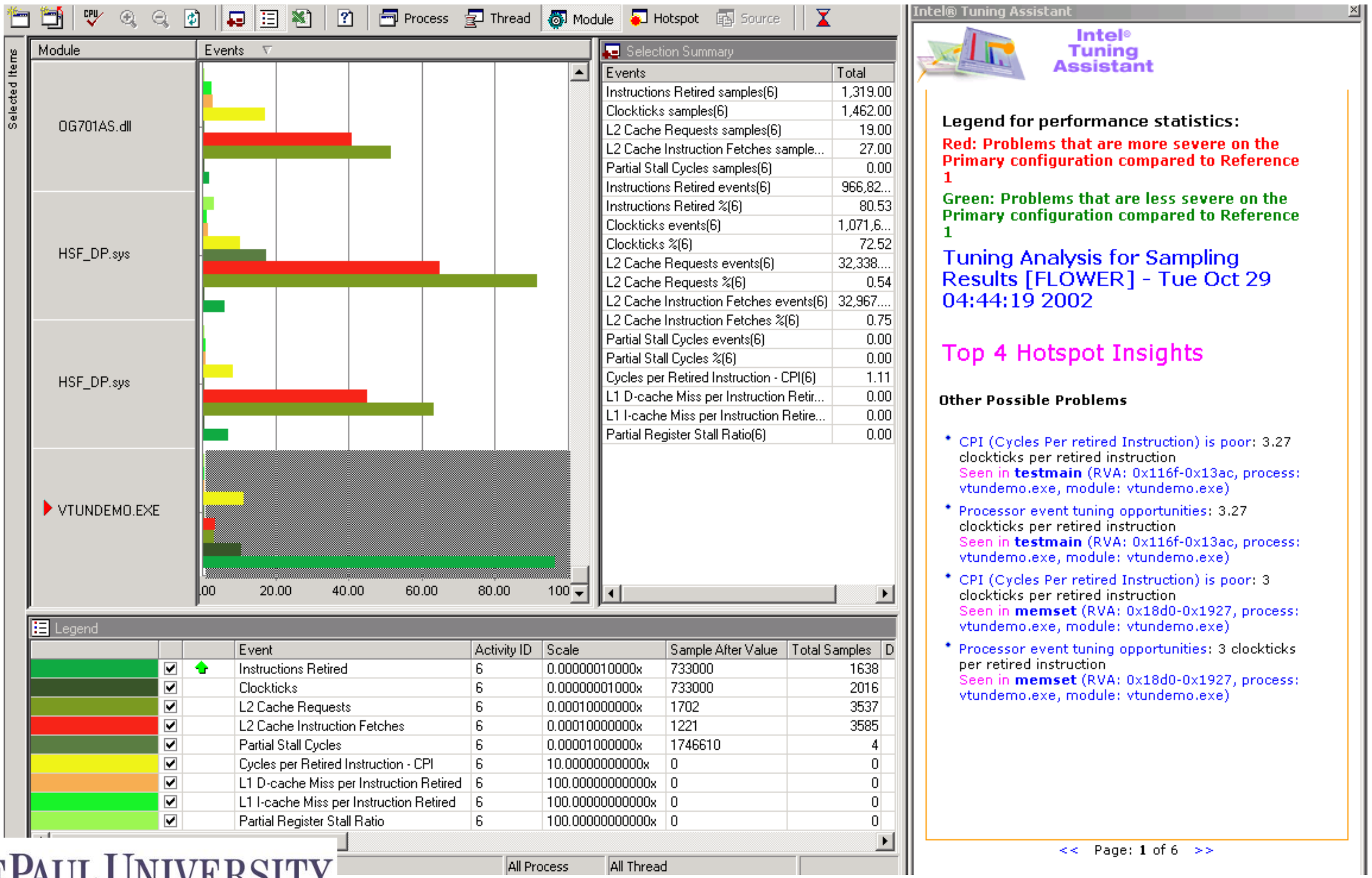
- ***Flat profiler***
  - Flat profilers compute the average call times, from the calls, and do not break down the call times based on the callee or the context.
- ***Call-graph profiler***
  - Call graph profilers show the call times, and frequencies of the functions, and also the call-chains involved based on the callee. However context is not preserved.

Profilers

- Helps you identify and characterize performance issues by:
  - Collecting performance data from the system running your application.
  - Organizing and displaying the data in a variety of interactive views, from system-wide down to source code or processor instruction perspective.
  - Identifying potential performance issues and suggesting improvements.

VTune: Example

VTune
Sampling Data



Events	Total
Instructions Retired samples(6)	1,319.00
Clockticks samples(6)	1,462.00
L2 Cache Requests samples(6)	19.00
L2 Cache Instruction Fetches sample...	27.00
Partial Stall Cycles samples(6)	0.00
Instructions Retired events(6)	966,82...
Instructions Retired %(6)	80.53
Clockticks events(6)	1,071,6...
Clockticks %(6)	72.52
L2 Cache Requests events(6)	32,338....
L2 Cache Requests %(6)	0.54
L2 Cache Instruction Fetches events(6)	32,967....
L2 Cache Instruction Fetches %(6)	0.75
Partial Stall Cycles events(6)	0.00
Partial Stall Cycles %(6)	0.00
Cycles per Retired Instruction - CPI(6)	1.11
L1 D-cache Miss per Instruction Retir...	0.00
L1 I-cache Miss per Instruction Retire...	0.00
Partial Register Stall Ratio(6)	0.00

			Event	Activity ID	Scale	Sample After Value	Total Samples	D
🟩	✓	⬆	Instructions Retired	6	0.00000010000x	733000	1638	
⬛	✓		Clockticks	6	0.00000001000x	733000	2016	
🟩	✓		L2 Cache Requests	6	0.00010000000x	1702	3537	
🟥	✓		L2 Cache Instruction Fetches	6	0.00010000000x	1221	3585	
🟩	✓		Partial Stall Cycles	6	0.00001000000x	1746610	4	
🟨	✓		Cycles per Retired Instruction - CPI	6	10.00000000000x	0	0	
🟧	✓		L1 D-cache Miss per Instruction Retired	6	100.00000000000x	0	0	
🟩	✓		L1 I-cache Miss per Instruction Retired	6	100.00000000000x	0	0	
🟩	✓		Partial Register Stall Ratio	6	100.00000000000x	0	0	

Legend for performance statistics:

Red: Problems that are more severe on the Primary configuration compared to Reference 1

Green: Problems that are less severe on the Primary configuration compared to Reference 1

Tuning Analysis for Sampling Results [FLOWER] - Tue Oct 29 04:44:19 2002

Top 4 Hotspot Insights

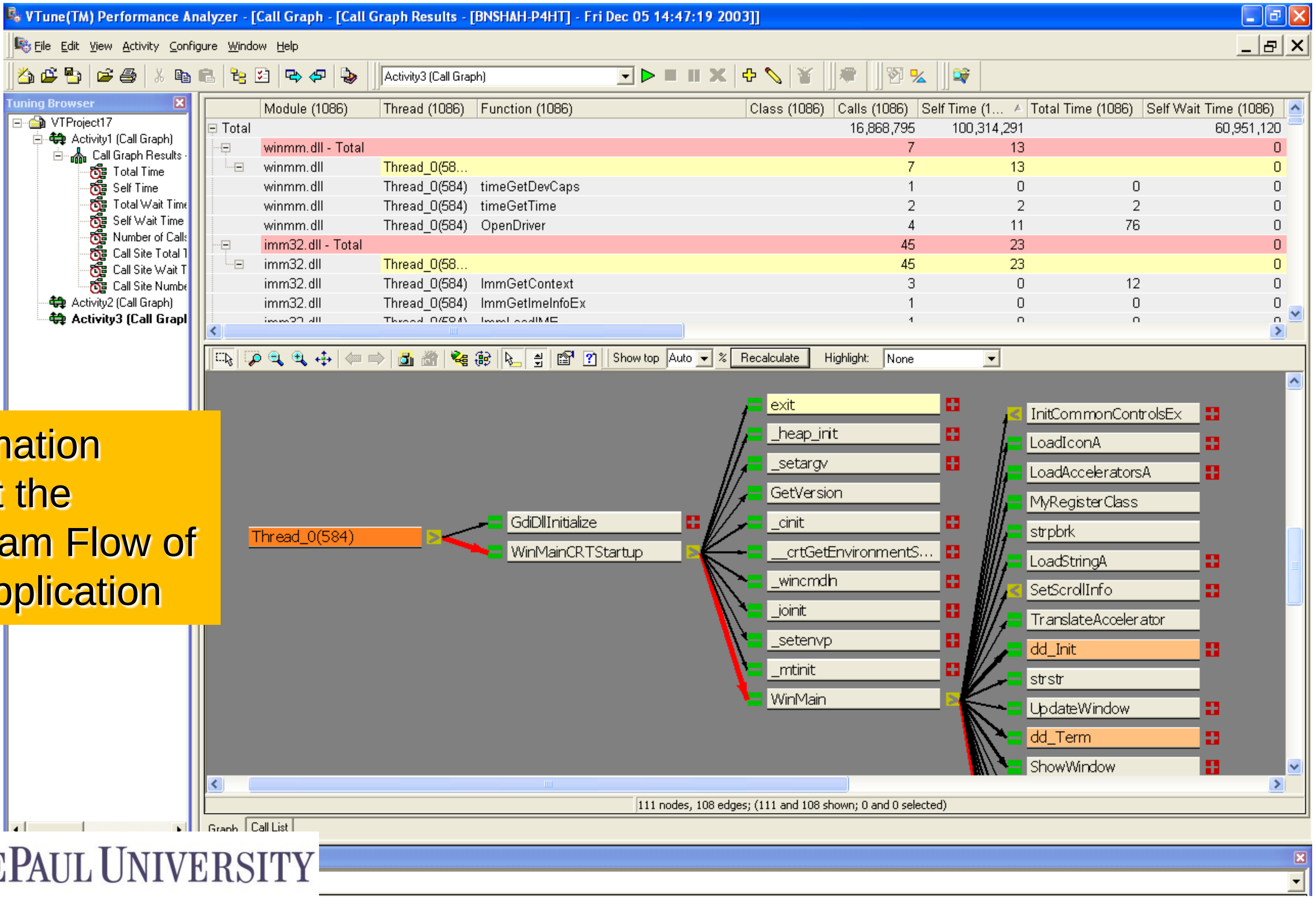
Other Possible Problems

- CPI (Cycles Per retired Instruction) is poor: 3.27 clockticks per retired instruction
  Seen in **testmain** (RVA: 0x116f-0x13ac, process: vtundemo.exe, module: vtundemo.exe)
- Processor event tuning opportunities: 3.27 clockticks per retired instruction
  Seen in **testmain** (RVA: 0x116f-0x13ac, process: vtundemo.exe, module: vtundemo.exe)
- CPI (Cycles Per retired Instruction) is poor: 3 clockticks per retired instruction
  Seen in **memset** (RVA: 0x18d0-0x1927, process: vtundemo.exe, module: vtundemo.exe)
- Processor event tuning opportunities: 3 clockticks per retired instruction
  Seen in **memset** (RVA: 0x18d0-0x1927, process: vtundemo.exe, module: vtundemo.exe)

<< Page: **1** of 6 >>

DePaul University

VTune
Call Graphs



VTune(TM) Performance Analyzer - [Call Graph - [Call Graph Results - [BNSHAH-P4HT] - Fri Dec 05 14:47:19 2003]]

File Edit View Activity Configure Window Help

Activity3 (Call Graph)

Tuning Browser
- VTProject17
  - Activity1 (Call Graph)
    - Call Graph Results
      - Total Time
      - Self Time
      - Total Wait Time
      - Self Wait Time
      - Number of Calls
      - Call Site Total T
      - Call Site Wait T
      - Call Site Numbe
  - Activity2 (Call Graph)
  - **Activity3 (Call Grapl**

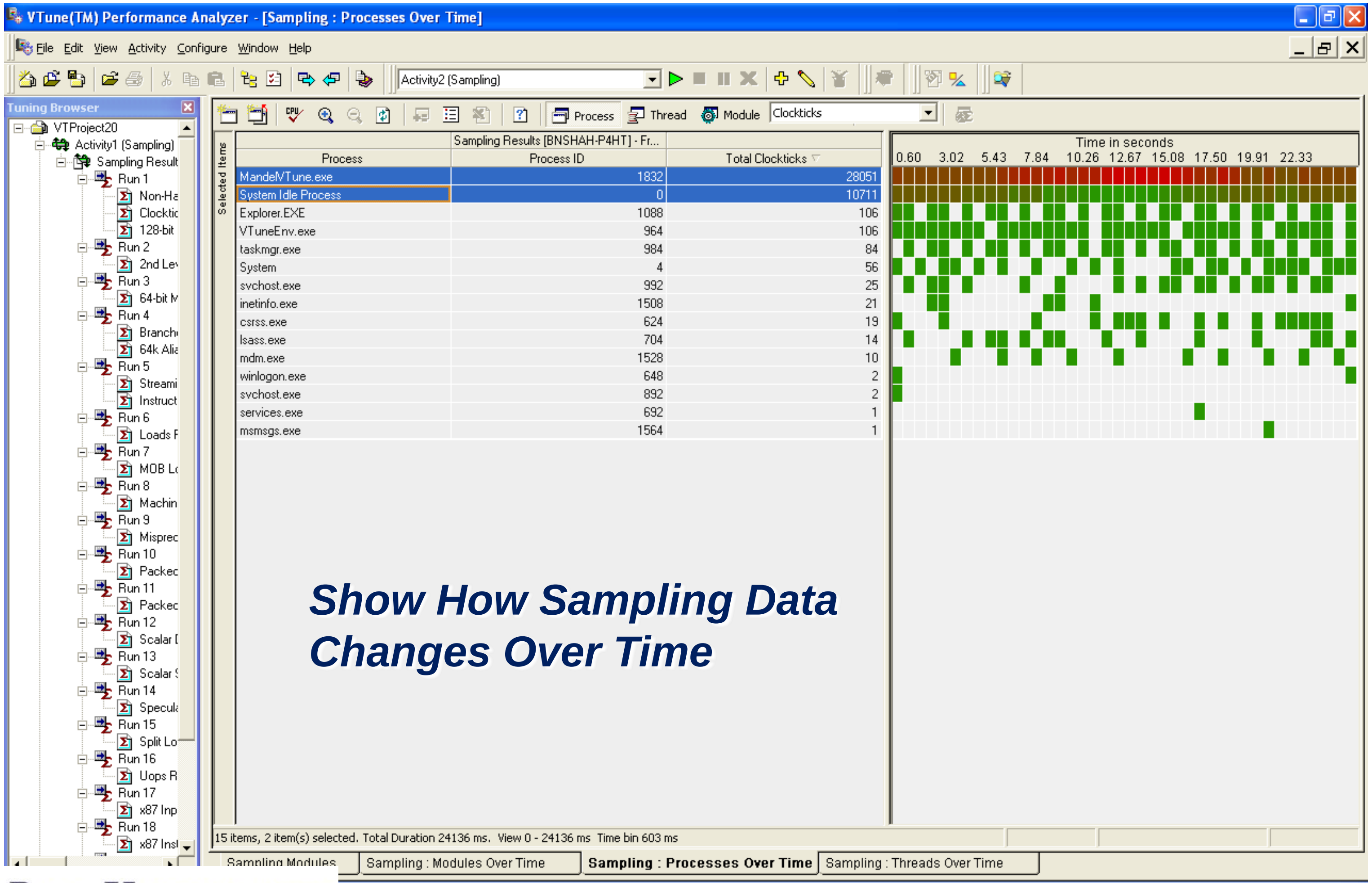
	Module (1086)	Thread (1086)	Function (1086)	Class (1086)	Calls (1086)	Self Time (1...	Total Time (1086)	Self Wait Time (1086)
Total					16,868,795	100,314,291		60,951,120
	winmm.dll - Total				7	13		0
	winmm.dll	Thread_0(58...			7	13		0
	winmm.dll	Thread_0(584)	timeGetDevCaps		1	0	0	0
	winmm.dll	Thread_0(584)	timeGetTime		2	2	2	0
	winmm.dll	Thread_0(584)	OpenDriver		4	11	76	0
	imm32.dll - Total				45	23		0
	imm32.dll	Thread_0(58...			45	23		0
	imm32.dll	Thread_0(584)	ImmGetContext		3	0	12	0
	imm32.dll	Thread_0(584)	ImmGetImeInfoEx		1	0	0	0

111 nodes, 108 edges; (111 and 108 shown; 0 and 0 selected)

Information About the Program Flow of the Application

DePaul University

VTune
Sampling over Time



VTune(TM) Performance Analyzer - [Sampling : Processes Over Time]

File Edit View Activity Configure Window Help

Activity2 (Sampling)

Tuning Browser

- VTProject20
  - Activity1 (Sampling)
    - Sampling Result
      - Run 1
        - Non-Ha
        - Clocktic
        - 128-bit
      - Run 2
        - 2nd Lev
      - Run 3
        - 64-bit M
      - Run 4
        - Branch
        - 64k Alia
      - Run 5
        - Streami
        - Instruct
      - Run 6
        - Loads F
      - Run 7
        - MOB Lo
      - Run 8
        - Machin
      - Run 9
        - Mispred
      - Run 10
        - Packed
      - Run 11
        - Packed
      - Run 12
        - Scalar I
      - Run 13
        - Scalar S
      - Run 14
        - Specula
      - Run 15
        - Split Lo
      - Run 16
        - Uops R
      - Run 17
        - x87 Inp
      - Run 18
        - x87 Inst

Process Thread Module Clockticks

Sampling Results [BNSHAH-P4HT] - Fr...

Process	Process ID	Total Clockticks
MandelVTune.exe	1832	28051
System Idle Process	0	10711
Explorer.EXE	1088	106
VTuneEnv.exe	964	106
taskmgr.exe	984	84
System	4	56
svchost.exe	992	25
inetinfo.exe	1508	21
csrss.exe	624	19
lsass.exe	704	14
mdm.exe	1528	10
winlogon.exe	648	2
svchost.exe	892	2
services.exe	692	1
msmsgs.exe	1564	1

Time in seconds: 0.60 3.02 5.43 7.84 10.26 12.67 15.08 17.50 19.91 22.33

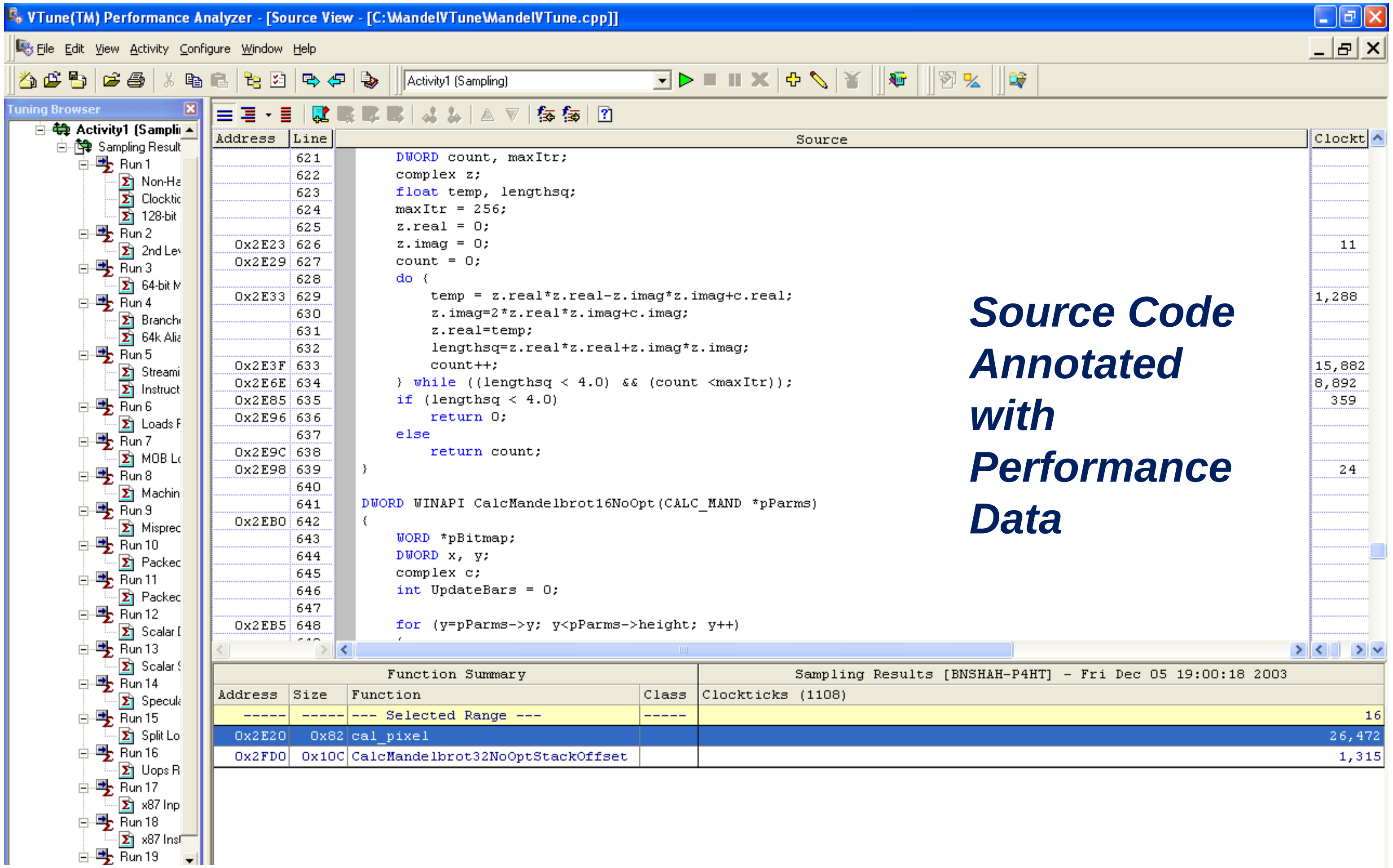
15 items, 2 item(s) selected. Total Duration 24136 ms. View 0 - 24136 ms Time bin 603 ms

Sampling Modules | Sampling : Modules Over Time | **Sampling : Processes Over Time** | Sampling : Threads Over Time

Show How Sampling Data Changes Over Time

DEPAUL UNIVERSITY

VTune
Source View



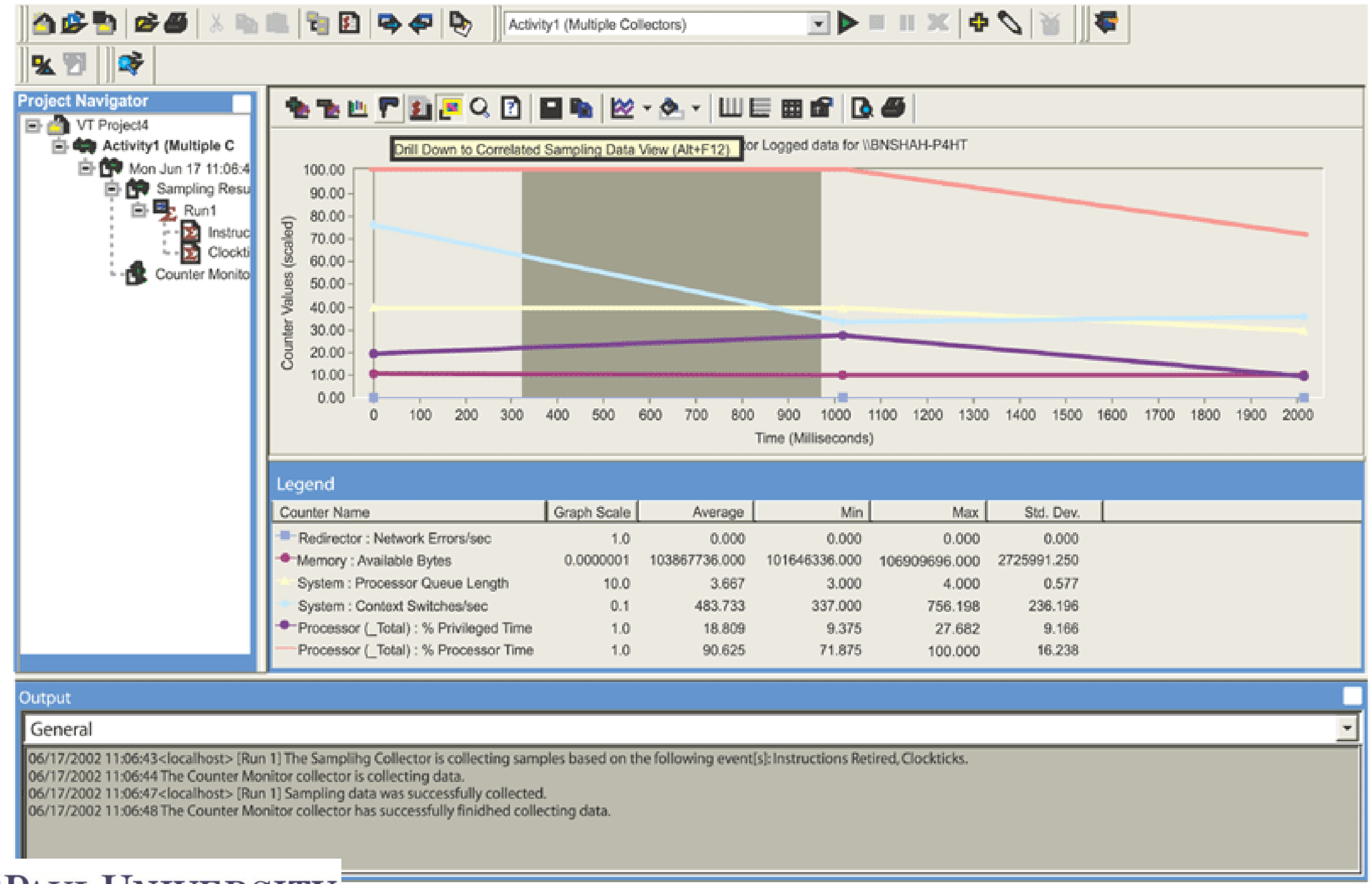
```
Address  Line   Source                                                          Clockt
         621        DWORD count, maxItr;
         622        complex z;
         623        float temp, lengthsq;
         624        maxItr = 256;
         625        z.real = 0;
0x2E23   626        z.imag = 0;                                                     11
0x2E29   627        count = 0;
         628        do {
0x2E33   629            temp = z.real*z.real-z.imag*z.imag+c.real;                1,288
         630            z.imag=2*z.real*z.imag+c.imag;
         631            z.real=temp;
         632            lengthsq=z.real*z.real+z.imag*z.imag;
0x2E3F   633            count++;                                              15,882
0x2E6E   634        } while ((lengthsq < 4.0) && (count <maxItr));            8,892
0x2E85   635        if (lengthsq < 4.0)                                         359
0x2E96   636            return 0;
         637        else
0x2E9C   638            return count;
0x2E98   639    }                                                                 24
         640
         641    DWORD WINAPI CalcMandelbrot16NoOpt(CALC_MAND *pParms)
0x2EB0   642    {
         643        WORD *pBitmap;
         644        DWORD x, y;
         645        complex c;
         646        int UpdateBars = 0;
         647
0x2EB5   648        for (y=pParms->y; y<pParms->height; y++)
```

Function Summary				Sampling Results [BNSHAH-P4HT] - Fri Dec 05 19:00:18 2003
Address	Size	Function	Class	Clockticks (1108)
-----	-----	--- Selected Range ---	-----	16
0x2E20	0x82	cal_pixel		26,472
0x2FD0	0x10C	CalcMandelbrot32NoOptStackOffset		1,315

Source Code Annotated with Performance Data

VTune

Counter Monitor



Legend

Counter Name	Graph Scale	Average	Min	Max	Std. Dev.
Redirector : Network Errors/sec	1.0	0.000	0.000	0.000	0.000
Memory : Available Bytes	0.0000001	103867736.000	101646336.000	106909696.000	2725991.250
System : Processor Queue Length	10.0	3.667	3.000	4.000	0.577
System : Context Switches/sec	0.1	483.733	337.000	756.198	236.196
Processor (_Total) : % Privileged Time	1.0	18.809	9.375	27.682	9.166
Processor (_Total) : % Processor Time	1.0	90.625	71.875	100.000	16.238

Output

General

```
06/17/2002 11:06:43<localhost> [Run 1] The Samplihg Collector is collecting samples based on the following event[s]: Instructions Retired, Clockticks.
06/17/2002 11:06:44 The Counter Monitor collector is collecting data.
06/17/2002 11:06:47<localhost> [Run 1] Sampling data was successfully collected.
06/17/2002 11:06:48 The Counter Monitor collector has successfully finidhed collecting data.
```

DEPAUL UNIVERSITY

VTune
Hot Spots

- ***Where*** in an application or system there is a ***significant*** amount of ***activity***
  - ***Where***
    - address in memory => OS process => OS thread => executable file or module => user function (requires symbols) => line of source code (requires symbols with line numbers) or processor (assembly) instruction
  - ***Significant***
    - activity that occurs infrequently probably does not have much impact on system performance
  - ***Activity***
    - time spent or other internal processor event
    - Examples of other events: Cache misses, branch mispredictions, floating-point instructions retired, partial register stalls, and so on.

VTune
Sampling Collector

- Periodically interrupt the processor to obtain the execution context
  - Time-based sampling (TBS) is triggered by:
    - Operating system timer services
    - Every n processor clockticks
  - Event-based sampling (EBS) is triggered by processor event counter overflow
    - These events are processor-specific, like L2 cache misses, branch mispredictions, floating-point instructions retired, and so on

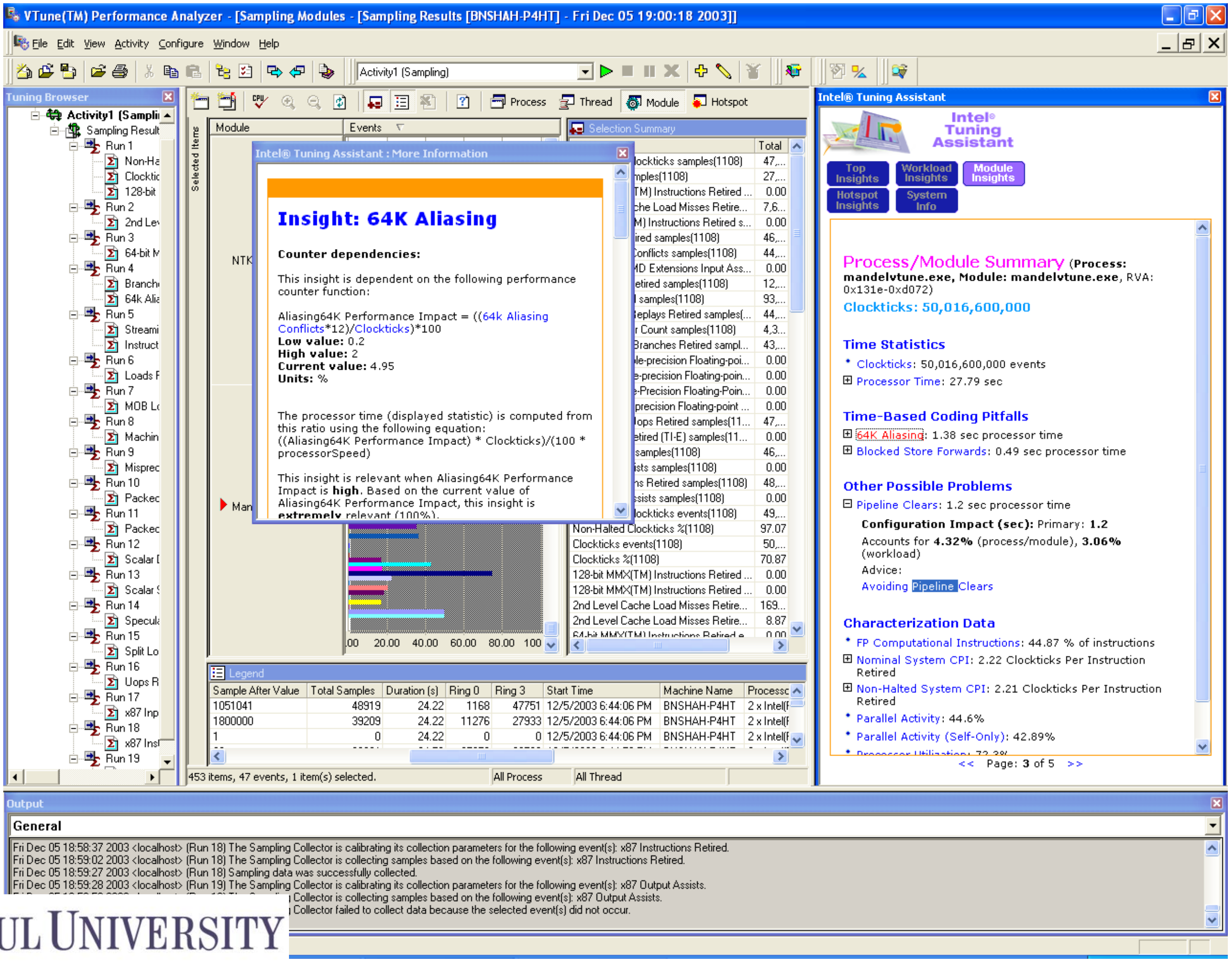
VTune
Benefits to Sampling

- You do not have to modify your code.
  - But DO compile/link with symbols and line numbers.
  - But DO make release builds with optimizations.
- Sampling is system-wide.
  - Not just YOUR application.
  - You can see activity in operating system code, including drivers.
- Sampling overhead is very low.
  - Validity is highest when perturbation is low.
  - Overhead can be reduced further by turning off progress meters in the user interface.

VTune
What Can You Profile?

- Win32 applications
- Stand-alone Win32* DLLs
- Stand-alone COM+ DLLs
- Java applications
- .NET* applications
- ASP.NET applications
- Linux32* applications

VTune
Tuning Assistant



VTune(TM) Performance Analyzer - [Sampling Modules - [Sampling Results [BNSHAH-P4HT] - Fri Dec 05 19:00:18 2003]]

Intel® Tuning Assistant : More Information

Insight: 64K Aliasing

Counter dependencies:

This insight is dependent on the following performance counter function:

Aliasing64K Performance Impact = ((64k Aliasing Conflicts*12)/Clockticks)*100
Low value: 0.2
High value: 2
Current value: 4.95
Units: %

The processor time (displayed statistic) is computed from this ratio using the following equation:
((Aliasing64K Performance Impact) * Clockticks)/(100 * processorSpeed)

This insight is relevant when Aliasing64K Performance Impact is **high**. Based on the current value of Aliasing64K Performance Impact, this insight is **extremely** relevant (100%).

Intel® Tuning Assistant

Process/Module Summary (**Process: mandelvtune.exe, Module: mandelvtune.exe**, RVA: 0x131e-0xd072)
Clockticks: 50,016,600,000

Time Statistics
- Clockticks: 50,016,600,000 events
- Processor Time: 27.79 sec

Time-Based Coding Pitfalls
- 64K Aliasing: 1.38 sec processor time
- Blocked Store Forwards: 0.49 sec processor time

Other Possible Problems
- Pipeline Clears: 1.2 sec processor time
  Configuration Impact (sec): Primary: **1.2**
  Accounts for **4.32%** (process/module), **3.06%** (workload)
  Advice:
  Avoiding Pipeline Clears

Characterization Data
- FP Computational Instructions: 44.87 % of instructions
- Nominal System CPI: 2.22 Clockticks Per Instruction Retired
- Non-Halted System CPI: 2.21 Clockticks Per Instruction Retired
- Parallel Activity: 44.6%
- Parallel Activity (Self-Only): 42.89%

<< Page: **3** of 5 >>

Sample After Value	Total Samples	Duration (s)	Ring 0	Ring 3	Start Time	Machine Name	Processo
1051041	48919	24.22	1168	47751	12/5/2003 6:44:06 PM	BNSHAH-P4HT	2 x Intel(F
1800000	39209	24.22	11276	27933	12/5/2003 6:44:06 PM	BNSHAH-P4HT	2 x Intel(F
1	0	24.22	0	0	12/5/2003 6:44:06 PM	BNSHAH-P4HT	2 x Intel(F

453 items, 47 events, 1 item(s) selected. All Process All Thread

Output — General

```
Fri Dec 05 18:58:37 2003 <localhost> (Run 18) The Sampling Collector is calibrating its collection parameters for the following event(s): x87 Instructions Retired.
Fri Dec 05 18:59:02 2003 <localhost> (Run 18) The Sampling Collector is collecting samples based on the following event(s): x87 Instructions Retired.
Fri Dec 05 18:59:27 2003 <localhost> (Run 18) Sampling data was successfully collected.
Fri Dec 05 18:59:28 2003 <localhost> (Run 19) The Sampling Collector is calibrating its collection parameters for the following event(s): x87 Output Assists.
Collector is collecting samples based on the following event(s): x87 Output Assists.
Collector failed to collect data because the selected event(s) did not occur.
```

Provides Tuning Advice Based on Performance Data

Popular Profilers

VTune

- Intel® VTune™ Amplifier XE is a software performance analysis for 32 and 64-bit x86 based machines, and has both GUI and command line interfaces.
  - OS: Linux and Microsoft Windows
- Intel VTune Amplifier XE assists in various profiling
  - including stack sampling,
  - thread profiling
  - hardware event sampling.
- Profiler result consists of details:
  - ***time spent*** in each sub routine which can be drilled down to the instruction level
  - time taken by the instructions are indicative of any stalls in the ***pipeline*** during instruction execution
  - tool can be also used to analyze ***thread*** performance.

Microsoft Visual Studio Team System Profiler

- Visual Studio Team System Profiler by Microsoft
  - Integrated into the Visual Studio Team System (VSTS) suite and the Development Edition of Visual Studio
- Works in either in ***sampling*** or ***instrumentation*** mode.
  - VSTS profiler helps to resolve performance problems in code written for the .NET platform or Visual C++ code
- Profiler the application will need to be launched and the component (screen) to be analyzed needs to be executed normally and exited
  - After completing, the profiler gives a summary of the elapsed time for each of the functions and the number of time each function is called. Also the memory usage data of the objects can also be tracked.

Apple: Instruments

- Instruments (formerly Xray)
  - Performance analyzer and visualizer, integrated in Xcode 3.0 and later
  - Instruments shows a time line displaying any event occurring in the applicatio
  - CPU activity variation, memory allocation, and network and file activity, together with graphs and statistics.
- Group of events are monitored via customizable "instruments“
  - Which record user generated events and replay (emulate) them exactly as many times as needed
  - Developer can see the effect of code changes without actually doing the repetitive work.
  - Instrument Builder feature allows the creation of custom analysis instruments.
- Built-in instruments can track
  - User events, such as keyboard keys pressed and mouse moves and clicks with exact time.
  - CPU activity of processes and threads.
  - Memory allocation and release, garbage collection and memory leaks.
  - File reads, writes, locks.
  - Network activity and traffic.
  - Graphics and inner workings of OpenGL.

You can get these tools

- Apple Instruments
  - Free part of Xcode
- VSTS profiler
  - MSDNAA DePaul for students
  - Ultimate Edition
- Intel VTune Profiler
  - Get a trial version today!

Thank You!

- Questions?

DePaul University

Character Strings

Optimized C++

Ed Keenan

DEPAUL UNIVERSITY

Goals

- Strings
  - Historical
    - PIA - Background

Last class?

DePaul University

References:

- Internet articles,
  - *"The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)"*
  - *"Back to Basics"*
    - by Joel Spolsky
      - http://www.joelonsoftware.com/
  - This Guy is Great!
- He has a great book
  - *"Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity "*
    - Please buy a copy – endless enjoyment

DePaul University

Before we talk optimization

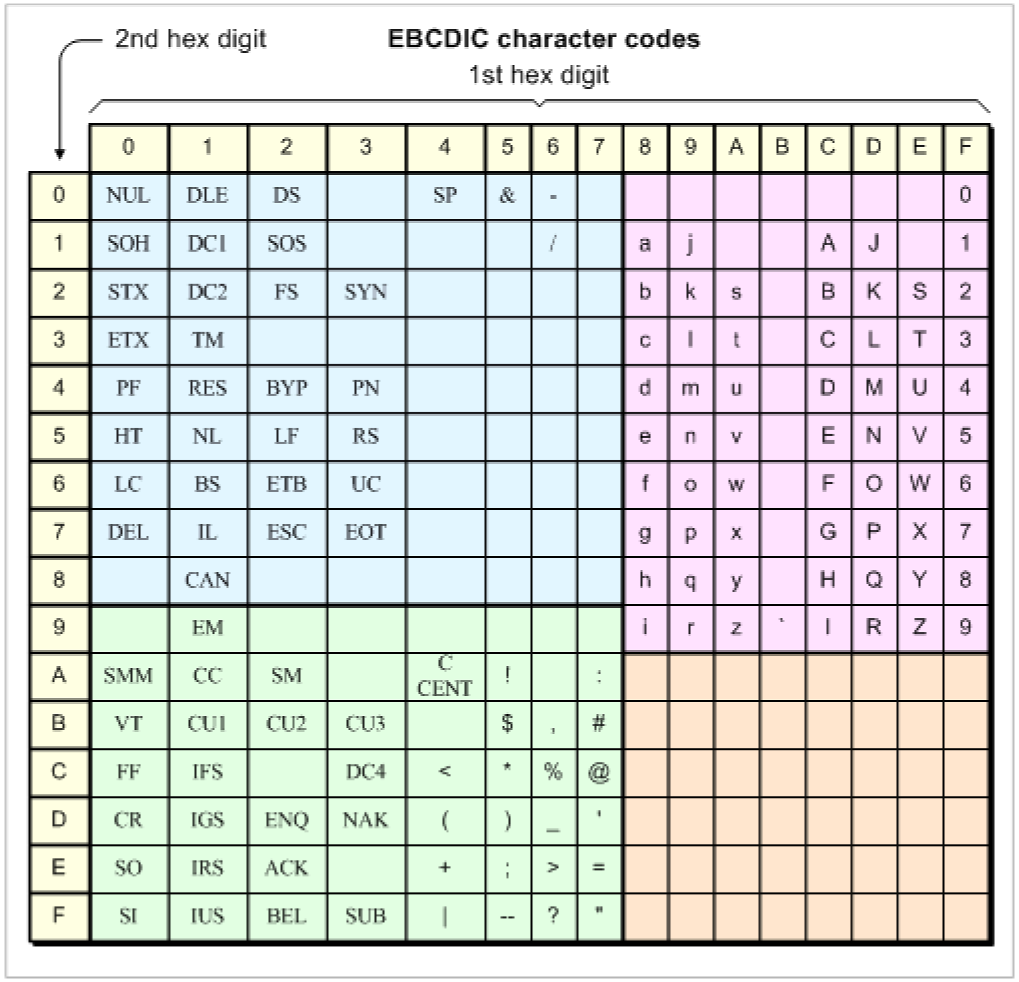
- We should define what a string is?
  - I thought this was a simple task
    - 1-2 slides tops
  - But hang-on
    - Bring a bag lunch
    - It’s a doozy
  - I learned a lot in this exploration
    - Hopefully you will.

History 1950-1960s

- EBCDIC
  - Extended Binary Coded Decimal Interchange Code (EBCDIC) is an 8-bit character encoding used mainly on IBM mainframe and IBM midrange computer operating systems.
  - EBCDIC descended from the code used with punched cards and the corresponding six bit binary-coded decimal code used with most of IBM's computer peripherals of the late 1950s and early 1960s
- BCD
  - What is that?

History 1950-1960s

- EBCDIC alphabetic characters follow a punched card encoding convention rather than a linear ordering like ASCII.
  - One consequence of this is that incrementing the character code for "I" does not produce the code for "J“
- Question:
  - The American government went to IBM to come up with an encryption standard, and they came up with…
- Answer:
  - EBCDIC!



2nd hex digit

EBCDIC character codes

1st hex digit

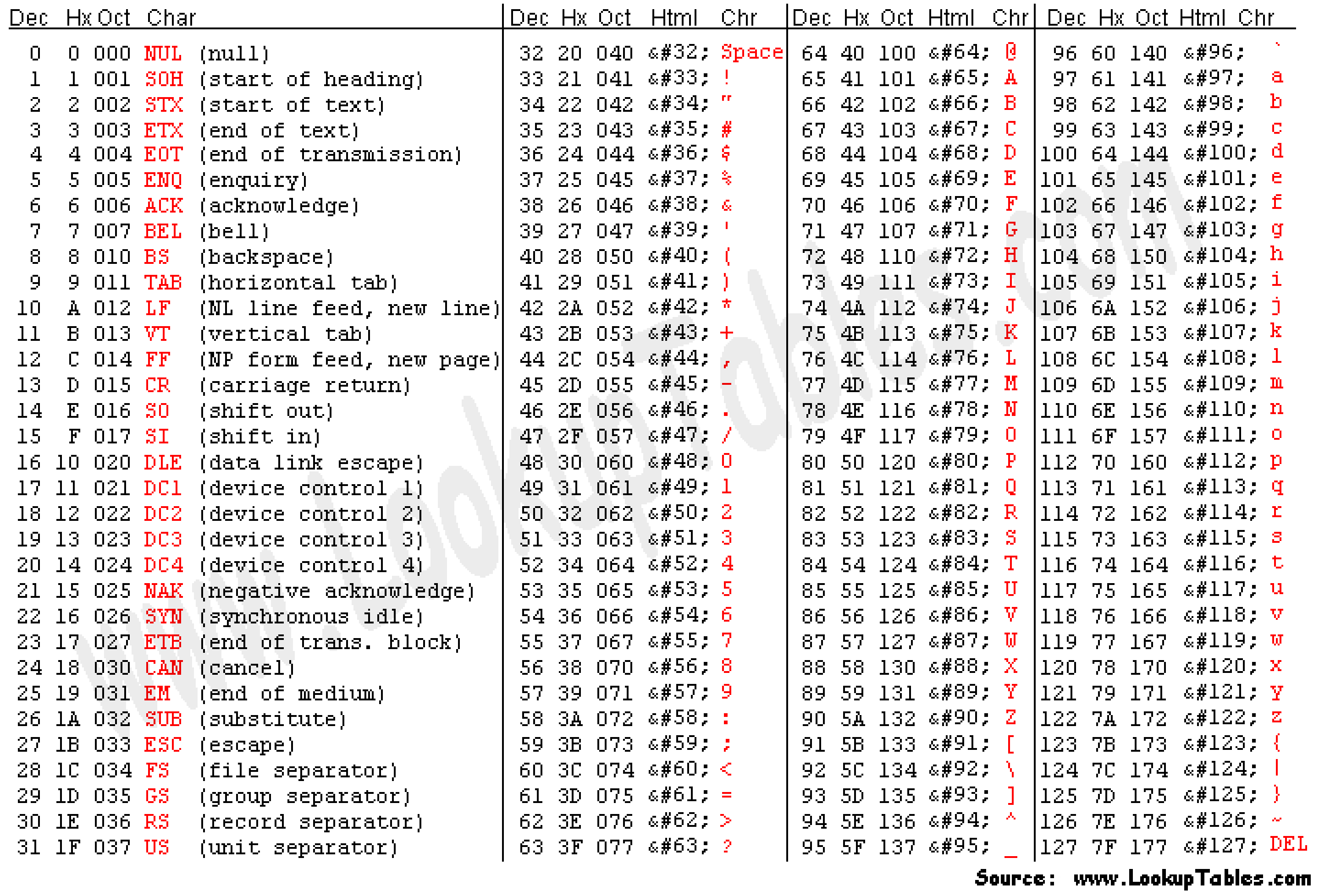
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	-									0
1	SOH	DC1	SOS				/		a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	TM							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z	`	I	R	Z	9
A	SMM	CC	SM		C CENT	!		:								
B	VT	CU1	CU2	CU3		$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	'								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	|	--	?	"								

DePaul University

Simple character set

- ASCII
  - Only characters that mattered were good old unaccented English letters
  - A code for them called ASCII which was able to represent every character using a number between 32 and 127
  - Everything be stored in 7 bits
    - Decimal (0 - 127)
    - Hex (0 – 0x3F)
- Example:
  - <space> was 32d
  - Letter “A’ was 65d
  - Letter “B” was 66d, etc.

ASCII (0-127)



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Opportunity?

- Most computers in those days were using 8-bit bytes
  - So *not only* could you store every possible ASCII character
  - Had a whole bit to spare.
  - 128 extra entries
- What do we do with all that extra space?

Parity

- Aside:
  - Early days data was encoded in 7 bits, the most significant bit was the parity bit
  - Used for error encoding
  - Hardware level assignment / check
- Result
  - 0 – if sum of bits 0:7 is even
  - 1 – if sum of bits 0:1 is odd

Parity differences

- Size:
  - 2 MB 3.5 DD DS floppy
    - Double density, double sided
    - IBM: 40 tracks, 9 sectors, 512 bytes per sector
  - Formatted for IBM:
    - 40 * 9 * 512 * 8 bits = 1474560 bits
    - 1474560 bits / (1024 * 1024) = 1.406 MB
  - Formatted for Mac
    - 40 * 9 * 512 * 8 * (64/56) parity ratio = 1685211 bits
    - 1685211 bits / (1024 * 1024) = 1.607 MB

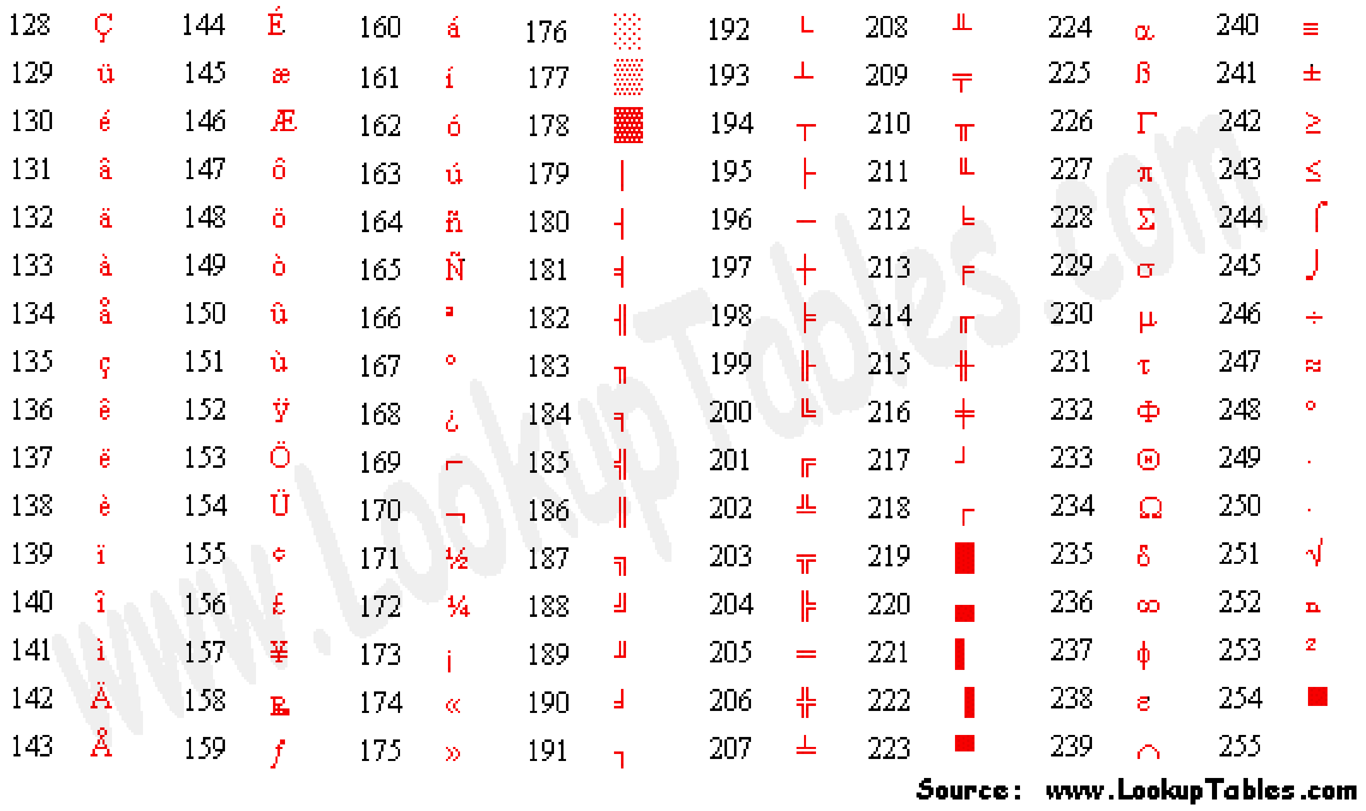
ASCII

- Unprintable code
  - Codes below 32 were called *unprintable*
  - Used for control characters
    - 7 which made your computer beep
    - 12 which caused the current page of paper to go flying out of the printer and a new one to be fed in

Using the extended range

- Use codes 128-255 for our own purposes?
  - Many people had their own ideas
  - IBM-PC
    - OEM character set which provided some accented characters for European languages
    - Added a bunch of line drawing characters
      - horizontal bars, vertical bars, horizontal bars with little dingle-dangles dangling off the right side, etc.,
      - Use these line drawing characters to make spiffy boxes and lines on the screen
      - Can still see on the 8088 computer at your dry cleaners

Extended ASCII (128-255)



128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

DePaul University

Wild west extended range

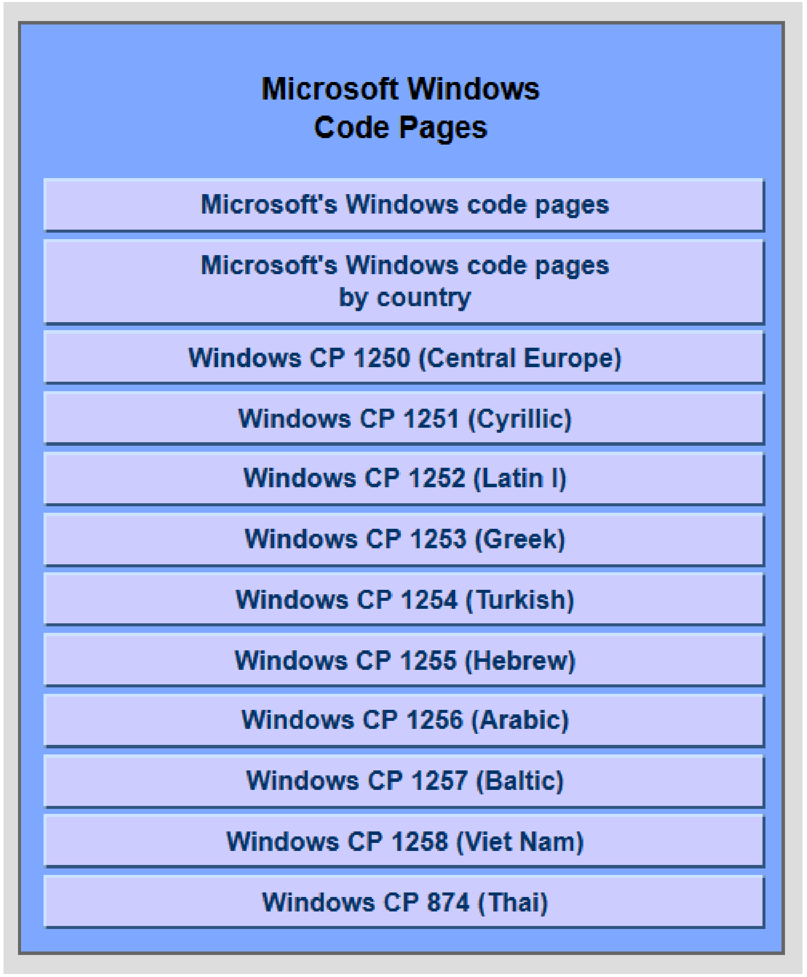
- Outside of America all kinds of different OEM character sets were created
  - All used the top 128 characters for their own purposes
  - Example:
    - character code 130 would display as ***é*** in the USA
    - Hebrew letter Gimel (***ג***) in Israel
      - Americans would send their ***résumés*** to Israel they would arrive as ***rגsumגs***
    - In Russian, there were lots of different ideas of what to do with the upper-128 characters, so you couldn't even reliably interchange Russian documents.

ANSI standard

- OEM free-for-all got codified in the ANSI standard
  - Codes < 128
    - Which was pretty much the same as ASCII
  - Codes >= 128 many different ways to handle characters
    - Different systems depending on its ***Code Page***
      - Israel - code page is 862
      - Greece – code page is 737
    - If they are below 128 uses the same character mapping
    - If greater than 128, characters are mapped based on its Code Page
- National versions of MS-DOS had dozens of these code pages
  - Everything from English to Icelandic
  - a few "multilingual" code pages that could do Esperanto and Galician *on the same computer.*

Code pages

- Code pages
  - http://www.i18nguy.com/unicode/codepages.html#msftdos
- Many different pages
  - Take a look



Microsoft Windows
Code Pages

Microsoft's Windows code pages

Microsoft's Windows code pages by country

Windows CP 1250 (Central Europe)

Windows CP 1251 (Cyrillic)

Windows CP 1252 (Latin I)

Windows CP 1253 (Greek)

Windows CP 1254 (Turkish)

Windows CP 1255 (Hebrew)

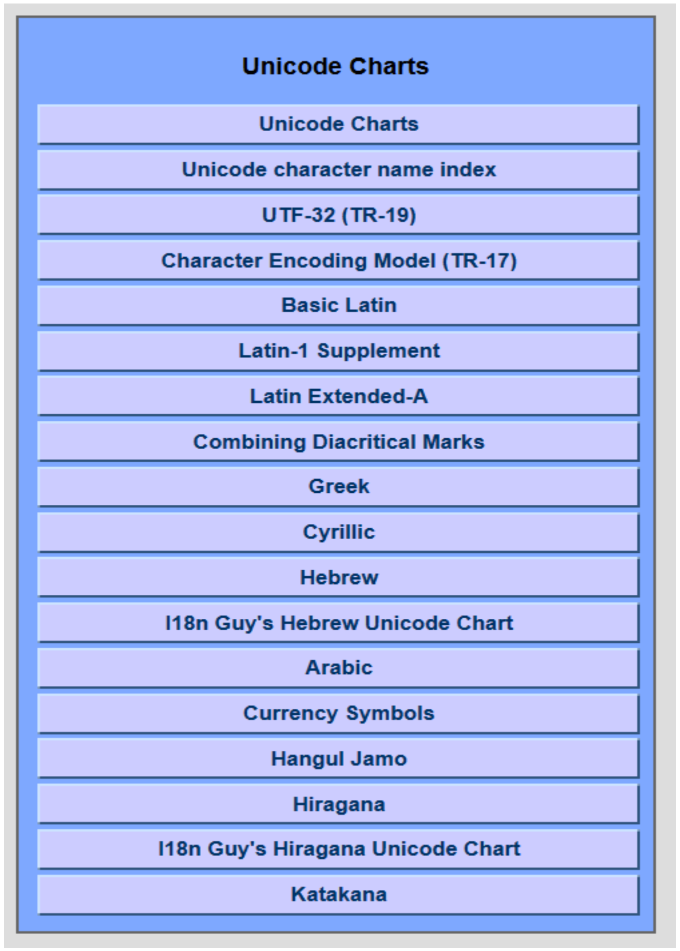
Windows CP 1256 (Arabic)

Windows CP 1257 (Baltic)

Windows CP 1258 (Viet Nam)

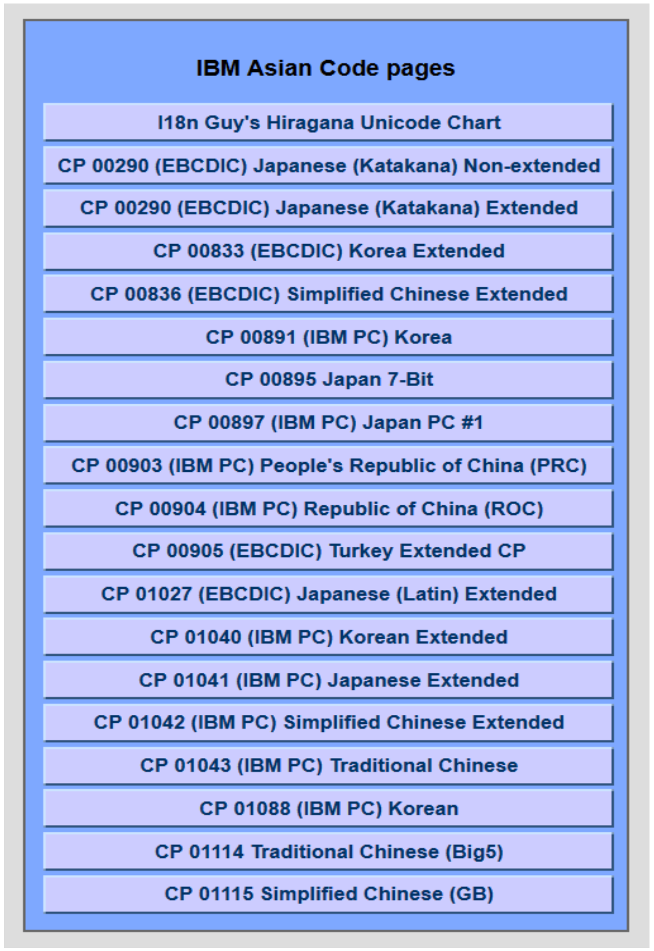
Windows CP 874 (Thai)

Code Pages



Unicode Charts

- Unicode Charts
- Unicode character name index
- UTF-32 (TR-19)
- Character Encoding Model (TR-17)
- Basic Latin
- Latin-1 Supplement
- Latin Extended-A
- Combining Diacritical Marks
- Greek
- Cyrillic
- Hebrew
- I18n Guy's Hebrew Unicode Chart
- Arabic
- Currency Symbols
- Hangul Jamo
- Hiragana
- I18n Guy's Hiragana Unicode Chart
- Katakana



IBM Asian Code pages

- I18n Guy's Hiragana Unicode Chart
- CP 00290 (EBCDIC) Japanese (Katakana) Non-extended
- CP 00290 (EBCDIC) Japanese (Katakana) Extended
- CP 00833 (EBCDIC) Korea Extended
- CP 00836 (EBCDIC) Simplified Chinese Extended
- CP 00891 (IBM PC) Korea
- CP 00895 Japan 7-Bit
- CP 00897 (IBM PC) Japan PC #1
- CP 00903 (IBM PC) People's Republic of China (PRC)
- CP 00904 (IBM PC) Republic of China (ROC)
- CP 00905 (EBCDIC) Turkey Extended CP
- CP 01027 (EBCDIC) Japanese (Latin) Extended
- CP 01040 (IBM PC) Korean Extended
- CP 01041 (IBM PC) Japanese Extended
- CP 01042 (IBM PC) Simplified Chinese Extended
- CP 01043 (IBM PC) Traditional Chinese
- CP 01088 (IBM PC) Korean
- CP 01114 Traditional Chinese (Big5)
- CP 01115 Simplified Chinese (GB)

DePaul University

Asia problem with ANSI

- Asia
  - Even more crazy things
  - Asian alphabets have thousands of letters
    - Never going to fit into 8 bits.
- DBCS - double byte character set
  - Solved this messy system.
  - Some letters were stored in one byte and others took two.

Moving down a DBCS string

- It was easy to move forward in a string
  - Very hard to move backwards due to the encoding.
- Programmers were encouraged ***NOT*** to use s++ and s-- to move forwards and backwards
  - Use Windows' *AnsiNext* and *AnsiPrev* which knew how to deal with the whole mess

But wait there’s more…

- Programmers just pretended that a byte was a character
  - A character was 8 bits
  - You never
    - moved a string from one computer to another
    - spoke more than one language
  - Magically all worked
- Internet became common place
  - To move strings from one computer to another
  - Whole mess came tumbling down

Introducing Unicode

- Unicode was a brave effort to create a single character set
  - Included every reasonable writing system
    - Including Klingon and Elvish
- Some people are under the **misconception** that Unicode is simply a 16-bit code
  - Each character takes 16 bits and therefore there are 65,536 possible characters
  - **→ This is not, actually, correct**
    - It is the single most common myth about Unicode
      - Don't feel bad.

DEPAUL UNIVERSITY

Unicode – letter definition

- Unicode has a different way of thinking about characters
  - you have to understand the Unicode way of thinking of things or nothing will make sense.
- We've assumed that a letter maps to some bits which you can store on disk or in memory:
    - A -> 0100 0001
- In Unicode
  - A letter maps to a **code point** which is still just a theoretical concept.
- In Unicode,
  - the letter A is an atomic individual unit
    - A
  - this letter A
    - Is different than B
    - Is different than a
    - Same as A and ***A*** and A.
  - The idea that A in a Times New Roman font is the same character as the A in a Helvetica font, but *different* from "a" in lower case

DEPAUL UNIVERSITY

Unicode – code point

- Every platonic letter in every alphabet is assigned a magic number by the Unicode consortium which is written like this: **U+0639**.
  - This magic number is called a ***code point***.
  - The U+ means "Unicode" and the numbers are hexadecimal.
  - **Example:**
    - **U+0639** is the Arabic letter Ain.
    - The English letter A would be **U+0041**.
  - Unicode website
    - http://www.unicode.org/
- No limit on the number of letters that Unicode can define
  - Gone beyond 65,536 so not every unicode letter can really be squeezed into 2 bytes
  - In fact some unicode is 6 bytes long

Unicode - Example

- String = "*Hello*"
  - in Unicode, corresponds to these five code points:
    - U+0048 U+0065 U+006C U+006C U+006F.
- Just a bunch of code points.
  - Numbers, really.
  - Now how is it stored in memory?

Unicode - Encodings

- The earliest idea for Unicode encoding
  - Led to the myth about the two bytes
    - Just store those numbers in two bytes each.
  - So “Hello” becomes
    - 00 48 00 65 00 6C 00 6C 00 6F
  - Couldn't it also be?:
    - 48 00 65 00 6C 00 6C 00 6F 00
- Early implementors wanted to be able to store their Unicode code points in big-endian **or** little-endian mode
  - Whichever their particular CPU was fastest at
  - Unicode was stored in *two* ways

Unicode – Byte Order Marks

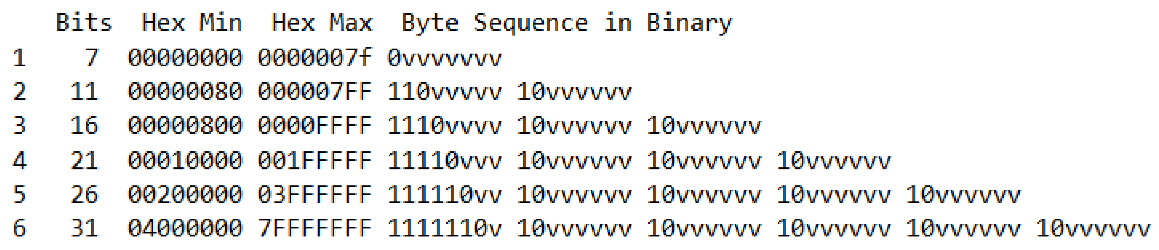
- Programmers were forced to come up with the bizarre convention of storing a 0xFEFF at the beginning of every Unicode string
  - Called a ***Unicode Byte Order Mark***
  - If you are swapping your high and low bytes it will look like a 0xFFFE
  - Programmer reading your string will know that they have to swap every other byte
- Heisenberg effect?
  - How do you know if you are reading a Little-Endian from a Big-Endian?

Unicode – All those 0s

- For a while it seemed like that might be good enough, but programmers were complaining.
  - "Look at all those zeros!“
  - Americans looking at English text which rarely used code points above U+00FF.
  - People wanted to *conserve bytes*
  - Most people decided to ignore Unicode for several years and in the meantime things got worse.

UTF-8

- UTF-8 was another system of Unicode code points
  - magic U+ numbers
  - memory using 8 bit bytes
  - http://www.utf-8.com
- In UTF-8
  - Every code point from 0-127 is stored in a single byte.
  - Only code points 128 and above are stored using 2, 3, in fact, up to 6 bytes.



	Bits	Hex Min	Hex Max	Byte Sequence in Binary
1	7	00000000	0000007f	0vvvvvvv
2	11	00000080	000007FF	110vvvvv 10vvvvvv
3	16	00000800	0000FFFF	1110vvvv 10vvvvvv 10vvvvvv
4	21	00010000	001FFFFF	11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
5	26	00200000	03FFFFFF	111110vv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv
6	31	04000000	7FFFFFFF	1111110v 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv

- Side effect that English text looks exactly the same in UTF-8 as it did in ASCII, so Americans don't even notice anything wrong.
  - Only the rest of the world has to jump through hoops. (Go USA!)

DePaul University

UTF-8

- **“Hello”** which was
  - U+0048 U+0065 U+006C U+006C U+006F
- UTF-8
  - stored as 48 65 6C 6C 6F
  - same as ASCII, and ANSI, and every OEM character set on the planet
- UTF-8 is ignorant old string-processing code that wants to use a single 0 byte as the null-terminator will not truncate strings

Choices so far

- Unicode - 3 ways of storing
  - 2 byte:
    - UCS-2 (unicode 2 bytes)
      - Big Endian
      - Little Endian
    - UTF-16 16 bit (2 byte)
  - UTF - 8

Many other Unicode types

- UTF-7
  - Like UTF-8 but guarantees that the high bit will always be zero
  - Compatible to the old email system that thinks 7 bits are enough
- UCS-4,
  - Stores each code point in 4 bytes
  - Uniform size of each code point
    - A lot of wasted 0 for ASCII type of letters

Need to know its Encoding

- Processing a string in the wrong encoding
  - Yields ***?*** in the string
  - 100s of types of encoding
- Single Most Important Fact About Encodings
  - Please remember one extremely important fact:
    - ***It does not make sense to have a string without knowing what encoding it uses***
  - You can no longer stick your head in the sand and pretend that "plain" text is ASCII

Encoding declaration

- How do we preserve this information about what encoding a string uses?
  - For an email message
    - Expected to have a string in the header
      - *Content-Type: text/plain; charset="UTF-8"*
  - Similar idea for web pages

Missing Encoding declaration

- Problems with web pages/applications not encoding declaration
  - Gluing many web pages together
  - Many authors using front page generation
- Several internet browsers Guess
  - They look at the code points
  - Try to figure out a pattern
  - Determine a best guess to the encoding
    - Say Bulgaria or Korean

Windows wchar_t

- In C++ code we just declare strings as ***wchar_t*** ("wide char") instead of *char*
- Use the ***wcs*** functions instead of the *str* functions
    - Example:
        - ***wcscat*** and ***wcslen*** instead of *strcat* and *strlen*
- To create a literal UCS-2 string in C code you just put an L before it as so: ***L"Hello"***

OK – GOT it?

- We now understand
  - Formats
  - Types of strings

Way Strings work

- C Strings consist of a bunch of bytes followed by a null character, which has the value 0.
  - Implications:
    - There is no way to know where the string ends.
      - Need to moving through the string, looking for the null character at the end.
    - Your string can't have any zeros in it.
      - So an arbitrary binary blob like a JPEG picture CANNOT be in a C string.
- Why do C strings work this way?
  - Because the PDP-7 microprocessor
  - UNIX and the C programming language were invented
    - ASCIZ string meant "ASCII with a Z (zero) at the end."

Strcat

- Code for **strcat**
  - Function which appends one string to another.

```
void strcat( char* dest, char* src )
    {
        while (*dest) dest++;
        while (*dest++ = *src++);
    }
```

- we're walking through the first string looking for its null-terminator
- when we find it
  - we walk through the second string
  - Copying one character at a time onto the first string

Problem

- Append several strings together

```
char bigString[1000];   /* I never know how much to allocate...
bigString[0] = '\0';
strcat( bigString, "John, ");
strcat( bigString, "Paul, ");
strcat( bigString, "George, ");
strcat( bigString, "Joel ");
```

Shlemiel Painter’s Algo.

- 1st Day
  - Shlemiel gets a job as a street painter, painting the dotted lines down the middle of the road.
  - On the first day he takes a can of paint out to the road and finishes 300 yards of the road.
  - "That's pretty good!" says his boss, "you're a fast worker!" and pays him a kopeck.
- 2nd Day
  - The next day Shlemiel only gets 150 yards done.
  - "Well, that's not nearly as good as yesterday, but you're still a fast worker. 150 yards is respectable," and pays him a kopeck.
- 3rd Day
  - The next day Shlemiel paints 30 yards of the road. "Only 30!" shouts his boss.
  - "That's unacceptable! On the first day you did ten times that much work! What's going on?“
- Shlemiel
  - "I can't help it," says Shlemiel.
  - "Every day I get farther and farther away from the paint can!"

Strcat - Shlemiel

- Joke illustrates ***strcat***
  - Scan through the destination string every time
    - Looking for the null terminator again and again.
  - This function is much slower than it needs to be and doesn't scale well at all.
- Lots of code contain this problem.
  - Many file systems are implemented in a way that it's a bad idea to put *too many files in one directory*
  - Performance starts to drop off dramatically when you get thousands of items in one directory

Find the Painter’s Algo.

- There must be a Shlemiel the Painter's Algorithm in there somewhere.
- Whenever something seems like it should have linear performance but it seems to have n-squared performance
  - Look for hidden Shlemiels.
  - Often hidden by your libraries
  - Looking at a column of **strcats** or a **strcat** in a loop doesn't exactly shout out "n-squared," but that is what's happening.

Custom strcat

```
char* mystrcat( char* dest, char* src )
    {
        while (*dest) dest++;
        while (*dest++ = *src++);
        return --dest;
    }
```

- What have we done here?
  - At very little extra cost we're returning a pointer to the *end* of the new, longer string.
  - That way the code that calls this function can decide to append further without rescanning the string:

Rework the problem

```
char bigString[1000];     /* I never know how much to allocate... */
char *p = bigString;
bigString[0] = '\0';
p = mystrcat (p, "John, ");
p = mystrcat( p, "Paul, ");
p = mystrcat( p, "George, ");
p = mystrcat( p, "Joel ");
```

- Linear in performance
  - Not n-squared
  - Doesn't suffer from degradation when you have a lot of stuff to concatenate.

DePaul University

Pascal Strings

- Pascal Strings "fixed" stores byte count in first byte
  - They can contain zeros and are not null terminated.
    - Because a byte can only store numbers between 0 and 255
  - Pascal strings
    - Limited to 255 bytes in length
    - Not null terminated
    - Same amount of memory as ASCIZ strings
  - No looping just to figure out the length of your string.
    - It is monumentally faster.
- Used outside of Pascal
  - Old Macintosh OS used Pascal strings everywhere.
  - Many C programs used Pascal strings for speed.
  - Excel uses Pascal - Excel are limited to 255 bytes
    - One reason Excel is fast.

Create Pascal Strings

- Create a Pascal string literal in your C code, you had to write:

```
char* str = "\006Hello!";
```

- Had to count the bytes by hand, yourself, and hardcode it into the first byte of your string.
  - Lazy programmers would do this, and have slow programs:

```
char* str = "*Hello!";
str[0] = strlen(str) - 1;
```

- Notice in this case you've got a string that is null terminated (the compiler did that) as well as a Pascal string.
- These are called **fucked strings** because it's easier than calling them **null terminated pascal strings**

How much do we allocate?

```
char* bigString;
int i = 0;
i = strlen("John, ")
    + strlen("Paul, ")
    + strlen("George, ")
    + strlen("Joel ");
bigString = (char*) malloc (i + 1);
/* remember space for null terminator! */
```

- We have to scan through all the strings once
  - Figuring out how big they are
  - Scan through them again concatenating
- if you use Pascal strings the strlen operation is fast.

DEPAUL UNIVERSITY

Better Strcmp?

- Walks two character strings at a time
    - One by one
        - Byte by Byte
    - Until it finds a difference or until it hits NULL
- Mechanism
    - char *p1, char *p2

Better Strcmp?

- What if you pad your strings to 4 bytes?
  - You can use Unsigned Integer compares
  - 4x times faster
  - Add up to 3 bytes more data per string.
  - Is it worth it?

File names with directories

- Where is the difference?
  - “C:\code\gam491\instructor\Basics8 - Templates\Basics8\Basics8\A.cpp”
  - “C:\code\gam491\instructor\Basics8 - Templates\Basics8\Basics8\B.cpp”
- After 63 bytes of comparison
  - We find a difference
  - A lot of compares to find the difference

Bias towards your difference

- Write the strings backwards
  - “ppc.A\8scisaB\8scisaB\setalpmeT – 8scisaB\rotcurtsni\194mag\edoc\:C”
  - “ppc.B\8scisaB\8scisaB\setalpmeT – 8scisaB\rotcurtsni\194mag\edoc\:C”
- After 5 bytes of comparison
  - We find a difference
  - 5 is better than 63

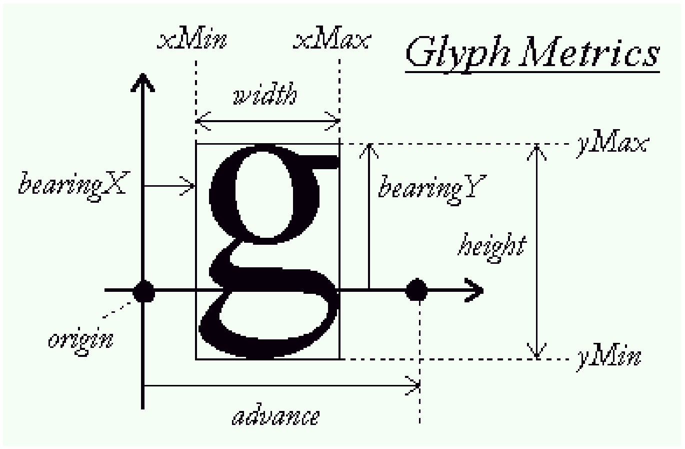
Strings?

- What if we HASHed the string?
  - If the strings are READ only
  - We are not mutating them
- Convert string to a 32 bit hash number
  - MD5, CRC32, etc
  - Use that as a lookup / compare
    - Find the associated string
- C# and Java use HASH tags for their strings

DePaul University

Glyphs

- Sometimes in Graphics we talk about strings when we really want Glyphs
- Glyphs – sprites on a 2D texture sheet that are cut and pasted together



Thank You!

- Questions?

DePaul University