Language Delay
Sean Parent | Principal Scientist
Vocalization
Vocalization

- Java was introduced in 1995 as a way to increase programmer productivity
  - Safe & secure
  - Simplified language with garbage collector
  - Portable
  - Object Oriented

- C++ (introduced in 1985) remained focused on performance and staying close to the hardware
  - Stay as efficient and as portable as C
  - Zero-overhead principle
  - Multi-paradigm
“Webpages that are optimized for Safari on iOS display and operate as designed (with the exception of any elements that rely on unsupported technologies, such as plug-ins, Flash, and Java).” – Apple
Java performance

About 297,000,000 results (0.33 seconds)
C++ on developer productivity
# C++

```cpp
struct _LIBCPP_VISIBLE piecewise_construct_t {
};

extern const piecewise_construct_t piecewise_construct; //= piecewise_construct_t();

template <class _T1, class _T2>
struct _LIBCPP_VISIBLE pair
{
    typedef _T1 first_type;
    typedef _T2 second_type;

    _T1 first;
    _T2 second;

    // pair(const pair&) = default;
    // pair(pair&&) = default;

    _LIBCPP_INLINE_VISIBILITY pair() : first(), second() {}

    _LIBCPP_INLINE_VISIBILITY
    pair(const _T1& __x, const _T2& __y) :
    first(__x), second(__y) {}

    template<class _U1, class _U2>
    _LIBCPP_INLINE_VISIBILITY
    pair(const pair<_U1, _U2>& __p)
    : first(__p.first), second(__p.second) {}

    _LIBCPP_INLINE_VISIBILITY
    pair(const pair& __p)
    NOEXCEPT (is_nothrow_copy_constructible<first_type>::value &&
```
The compiler provided the copy and move constructors
“We’re getting an error that has something to do with rvalue references and std::pair.”
template<class U, class V> pair(U&& x, V&& y);

- A pair<T, bool> was being constructed as “make_pair(x, false)”
- And generating a warning that an int was being converted to a bool...
- How?
Vocalization

ADMStandardTypes.h: #define false 0
AGFConvertUTF.cpp: #define false 0
ASBasic.h: #define false 0
ASBasicTypes.h: #define false 0
ASNumTypes.h: #define false 0
ASTypes.h: #define false 0
basics.h: #define false ((Bool32) 0)
common.h: #define false 0
config_assert.h: #define false 0
ConvertUTF.cpp: #define false 0
CoreExpT.h: #define false 0
ICCUtils.h: #define false 0
isparameter.cpp: #define false 0
PITypes.h: #define false FALSE
piwinutl.h: #define false FALSE
PSSupportPITypes.h: #define false FALSE
stdbool.h: #define false FALSE
t_9_017.cpp: #define false 0
WinUtilities.h: #define false FALSE
• Insert your own beautiful code here.
What are we trying to say?
Verbalization
Adobe Revel
Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)

- OpenGL
- OpenCL
- CUDA
- Direct Compute
- C++ AMP
- DirectX
- Intrinsic
- Auto-vectorization
- OpenCL
- TBB
- GCD
- OpenMP
- C++11
- Straight C++

0 750 1500 2250 3000

GPU
Vectorization
Multi-thread
Scalar

(GFlops)
The hardware has changed dramatically in the last 20 years (when typically a single scaler CPU was 100% of the machine)

Languages are much the same
Two kinds of parallel

Functional

Data Parallel
Vectorization

- Intrinsics: great speed potential, but...

```c
__m128i vDst = _mm_cvttps_epi32(_mm_mul_ps(_mm_cvtepi32_ps(vSum0), vInvArea));
```

- Moving target: MMX, SSE, SSE2, SSE3, SSE 4.1, SSE 4.2, AVX, AVX2, AVX3

- Solutions:
  - Auto-vectorization  #pragma SIMD
  - CEAN  Dest[:1] += src[start:length] + 2;
  - OpenCL
Data Parallel: 300 : 1

Sequential: 1 : 10
Verbalization

- Typical object oriented paradigms of using shared references to objects breaks down in a massively parallel environment
- Sharing implies either single threaded
  - Or synchronization
Amdahl's Law

\[ \frac{1}{(1-P) + \frac{P}{N}} \]
To utilize the hardware we need a fundamentally different vocabulary
  - Functional? Declarative? Reactive?

So far the only solutions that unlock the hardware are primitive and proprietary
  - Typically some form of constrained C-like language
Without addressing vectorization, GPGPU, and scalable parallelism, mainstream languages are just a scripting system to get to the other 99% of the machine through other specialized languages and libraries.

Common languages don't provide the words we need to verbalize.
What are we trying to express?
Oration
Content Ubiquity

- Ubiquitous access to:
  - calendar
  - contacts
  - notes & tasks
  - e-mail (corporate and personal)
  - A full web experience
  - Music
    - iTunes Music Match
    - Spotify
    - Pandora
  - Movies
    - Netflix
    - Vudu

- Photos
  - Flickr
  - Facebook
  - Adobe Revel

- Documents
  - Google Docs
  - Microsoft Office
  - Evernote

- Everything...
Content ubiquity is access to all your information, on all your devices, all of the time
The Problem

- Ubiquity has gone mainstream
  - A typical US household now has 3 TVs, 2 PCs, and 1 Smartphone
    - 1 in 3 households has an internet connected TV
    - A typical US worker has access to a PC at work or is provided an e-mail solution for communication
- The deluge of digital information has become a challenge to manage
  - How do I get this contract to my phone?
  - How do I get this video from my phone to my PC?
  - Which computer has the latest version of this photo?

Content ubiquity has become the expectation
The Technology is Here Now

- ≥ 3mbps broadband is available to 98% of the US population
- ≥ 3mbps mobile broadband is available to 99%
- US ranks 26th in broadband subscriptions per capita
  - Nearly every other tier one market is ahead of US
  - France (12), Germany (18), UK (19), Japan (27)
No Excuses

- Your data set is not too large
No Excuses

- Your application is not too interactive
Current hardware is capable enough

- Typical: 2x1GHz cores, 512GB RAM, 32GB SSD, GPU, 802.11n
- Revel runs the entire ACR image pipeline on an iPad 1 (half the above capabilities)
The Players

iCloud

SkyDrive

Dropbox
The Opportunity

- Focus on content ubiquity
  - all your content, instantly, on any available device
  - zero management overhead

- Users don't want to care about “The Cloud,” users want their content
The Challenge

- Content Ubiquity isn't a feature you can bolt-on
  - Dropbox, and similar technologies that require management and synchronization aren't the solution

- Achieving a seamless experience requires rethinking...
  - data model to support incremental changes
  - transactional models to support dynamic mobile environment
  - editor model to support partial editing (proxies, pyramid)
  - UI model to support touch, small devices, 10 foot interfaces
  - heavily asynchronous environment dealing with (relatively) high network latencies and trying to achieve zero latency on UI
Content Ubiquity Opens the Door to Sharing and Collaboration

- If you can make changes available to other devices immediately then you can make changes available to other apps immediately (works with sandboxing technology)

- If you can make documents available to all your devices then you can make documents available to others - supporting both collaboration and sharing
The Business

- In 2011 smartphone sales exceeded PC sales
- Tablets are expected to exceed PC sales by 2015
- There are 220M internet connected televisions
- Low margin, high volume
- Highly competitive, low barrier to entry

- Server and support costs dwarf development costs
Language Pain

- The market is very fragmented

### Desktop Operating System statistics on Net Applications

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7</td>
<td>44.48%</td>
</tr>
<tr>
<td>Windows XP</td>
<td>39.51%</td>
</tr>
<tr>
<td>OS X</td>
<td>7.08%</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>5.24%</td>
</tr>
<tr>
<td>Windows 8</td>
<td>2.26%</td>
</tr>
<tr>
<td>Linux</td>
<td>1.21%</td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>

**Desktop OS Market Share as of January 2013 Net Applications**

### Mobile Operating System statistics on Net Applications

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>60.56%</td>
</tr>
<tr>
<td>Android</td>
<td>24.51%</td>
</tr>
<tr>
<td>Java ME</td>
<td>9.32%</td>
</tr>
<tr>
<td>Symbian</td>
<td>1.78%</td>
</tr>
<tr>
<td>Blackberry</td>
<td>1.64%</td>
</tr>
<tr>
<td>Windows Phone</td>
<td>1.15%</td>
</tr>
<tr>
<td>Other</td>
<td>1.05%</td>
</tr>
</tbody>
</table>

**Mobile OS Market Share as of January 2013 Net Applications[^1]**
Language Pain

- To provide a solution requires you write for multiple platforms

- And many vendors are focusing on proprietary technology and languages

- Apple: Objective-C, Objective-C++, OpenCL, OpenGL, OpenGL ES
- Microsoft: C#, JavaScript, C++/CX, Direct-X, DirectCompute, C++AMP
- Google: Java, C/C++ through JNI, RenderScript Compute, OpenGL ES
- Browser: JavaScript, WebGL, C/C++ through Native Client (Chrome Only)
- NVIDIA: CUDA (OS X, Windows, Linux)
- Linux (Server Side): C/C++, Java, JavaScript, Python, Ruby, etc...
Platform Languages

Objective-C  Google  Java
Apple  Google
C++
Microsoft  C#  JavaScript
General Purpose-GPU Languages

- OpenCL*
  - Apple
- RenderScript
  - Google
- DirectCompute
  - Microsoft
Language Pain

- Vendor lock-in on commodity technologies only serves to slow development
  - including incorporating vendor specific technology that provides user benefit
Today it seems that we can only point and grunt
We need a common language to talk about systems with a vocabulary that scales to modern hardware.