



node.js[®]
INTERACTIVE

The image features the Node.js logo, which consists of the word "node" in a stylized, lowercase font where each letter is formed by a white geometric shape (a house-like shape for 'n', a hexagon for 'o', a hexagon with a dot for 'd', a hexagon with a dot for 'e', and a hexagon with 'JS' inside for 's'). To the right of the 's' is a registered trademark symbol (®). Below the logo, the word "INTERACTIVE" is written in a clean, white, uppercase, sans-serif font. The background is a vibrant cityscape at sunset, with a river in the foreground and a bridge on the right.



V8 engine of Node.js on IA: JavaScript-JITTED x86 machine code mapping profiling support and X87 Quark processor enabling

chunyang.dai@intel.com, Intel Company

Agenda

- **JavaScript-JITTED x86 machine code mapping profiling support in VTune**
 - Background: Scan of current profiling tools for Node.js
 - What's Intel VTune and and the JavaScript code / machine code mapping
 - How to use it in Node.js
- **X87 Quark processor enabling for V8 / Node.js**
 - Background
 - Intel's effort for X87 Quark processor enabling
 - How to build Node.js for Quark processor
- Q / A?

Scan of Current Node.js profiling for JavaScript code

• Existing Hierarchy of Node.js profiling

- Node.js' C++ level profiling – any platform-dependent profiling tool works
- Node.js' V8 profiling with limited JavaScript function information
 - Pass “--prof” flag to node and generate the V8 log file
 - Provided function level profiling data.
 - No detailed profiling information for JavaScript code
 - “perf” and v8 “--ll_prof” support on Linux etc
 - Provided function level profiling data.
 - Provided machine code level profiling data
 - No detailed profiling information for JavaScript code

```
[JavaScript]:
ticks total nonlib name
2065 10.9% 11.1% LazyCompile: *montReduce crypto.js:583:20
1484 7.8% 8.0% LazyCompile: *lin_solve navier-stokes.js:128:23
1307 6.9% 7.0% LazyCompile: *Scheduler.schedule richards.js:188:41
1058 5.6% 5.7% LazyCompile: *bnpSquareTo crypto.js:431:21
646 3.4% 3.5% LazyCompile: *one_way_unify1_nboyer earley-boyer.js:3635:37
620 3.3% 3.3% LazyCompile: *rewrite_nboyer earley-boyer.js:3604:30
523 2.8% 2.8% LazyCompile: *Plan.execute deltablue.js:773:35
464 2.4% 2.5% LazyCompile: *GeneratePayloadTree splay.js:49:29
414 2.2% 2.2% LazyCompile: *HandlerTask.run richards.js:465:38
405 2.1% 2.2% LazyCompile: *SplayTree.splay_ splay.js:293:38
323 1.7% 1.7% LazyCompile: *Loop2 earley-boyer.js:4272:65
301 1.6% 1.6% LazyCompile: *Loop3 earley-boyer.js:4286:85
286 1.5% 1.5% LazyCompile: *Flog.RayTracer.Shape.Sphere.intersect raytrace.js:426:24
270 1.4% 1.5% LazyCompile: *Flog.RayTracer.Engine.rayTrace raytrace.js:709:23
262 1.4% 1.4% LazyCompile: *Constraint.satisfy deltablue.js:172:41
246 1.3% 1.3% LazyCompile: *montMulTo crypto.js:606:19
244 1.3% 1.3% LazyCompile: *Flog.RayTracer.Vector.normalize raytrace.js:235:25
232 1.2% 1.3% LazyCompile: *exec native regexp.js:88:22
191 1.0% 1.0% LazyCompile: *deriv_trees earley-boyer.js:4254:43
180 0.9% 1.0% LazyCompile: *advect navier-stokes.js:204:20
176 0.9% 0.9% LazyCompile: *sc_loop1_98 earley-boyer.js:4258:251
143 0.8% 0.8% LazyCompile: *bnpFromString crypto.js:203:23
```

• A big missing

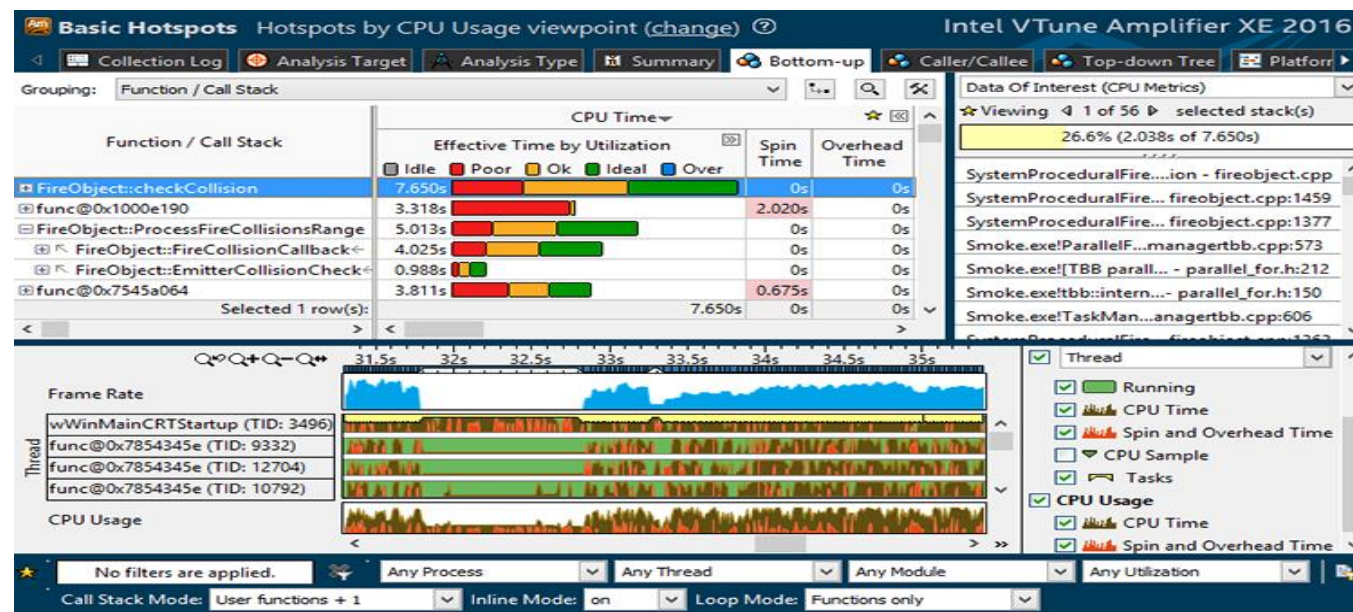
- Node.js' V8 profiling with detailed JavaScript source code mapping to JITTED assemble machine code

```
Ticks per symbol:
194079 9.6% LazyCompile:*montReduce crypto.js:583 [js]
0.02 0: 55 push ebp
0.01 1: 89 e5 mov ebp,esp
3: 56 push esi
0.01 4: 57 push edi
5: 83 ec 68 sub esp,0x68
8: c7 45 f4 00 00 00 00 mov DWORD PTR [ebp-0xc],0x0
f: 8b 45 fc mov eax,DWORD PTR [ebp-0x4]
12: 89 45 f0 mov DWORD PTR [ebp-0x10],eax
15: 89 c6 mov esi,eax
17: 3b 25 4c 87 ea 08 cmp esp,DWORD PTR ds:0x8ea874c
1d: 73 05 jae 0x24
1f: e8 bc 96 99 d4 call 0xd49996e0
24: 8b 5d 08 mov ebx,DWORD PTR [ebp+0x8]
27: f7 c3 01 00 00 00 test ebx,0x1
2d: 0f 84 37 80 f3 dd je 0xddf3806a
33: 81 7b ff 8d 16 f1 2b cmp DWORD PTR [ebx-0x1],0x2bfff1168d
0.02 3a: 0f 85 34 80 f3 dd jne 0xddf38074
40: 8b 7b 0b mov edi,DWORD PTR [ebx+0xb]
43: 89 7d ec mov DWORD PTR [ebp-0x14],edi
46: 8b 45 0c mov eax,DWORD PTR [ebp+0xc]
49: f7 c0 01 00 00 00 test eax,0x1
```

Intel VTune profiler and JITTED code mapping

- **What's Intel VTune profiler?**

- A commercial application for software performance analysis.
- It has both GUI and command line interfaces.
- It is available for Linux, Windows and Android operating systems.
- It provides functionality(API) to profile runtime generated code.
- It runs on top of Intel processors (please visit [Intel VTune Amplifier](#) for details)



- **VTune's JITTED code and source code mapping API:**

- Easy to use.
- Provide detailed dynamic generated code line <-> machine code level profiling/mapping capacity

Enable Node.js' JavaScript-JITTED x86 machine code mapping profiling support in VTune

- **Upstream Status:**

- This feature is landed in Node.js by pull request [#3785](https://github.com/nodejs/node/pull/3785).
- Thanks for Ben Noordhuis and others' timely review
- It should be available since Node.js V5.2.0.

- **How to build a Vtune-enable Node.js:**

- Step 1: Download Node.js code from github.
- Step 2: Compile Node.js with special flag: on Windows OS:

```
./vcbuild.bat --enable-vtune [other flags]
```

on Linux and android:

```
./configure --enable-vtune-profiling [other flags]
```

```
./make
```

src, build: Enable Intel Vtune profiling for JavaScript #3785

Closed cdai2 wants to merge 1 commit into nodejs:master from cdai2:master

Conversation 18 | Commits 1 | Files changed 4

cdai2 commented 18 days ago

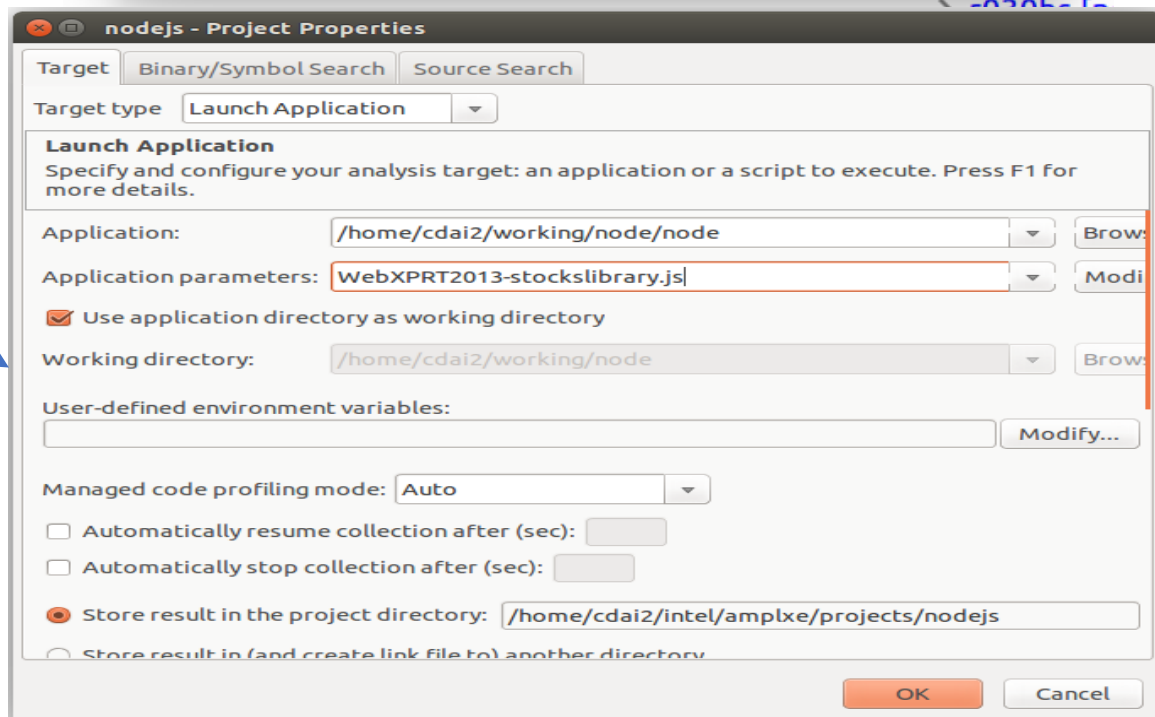
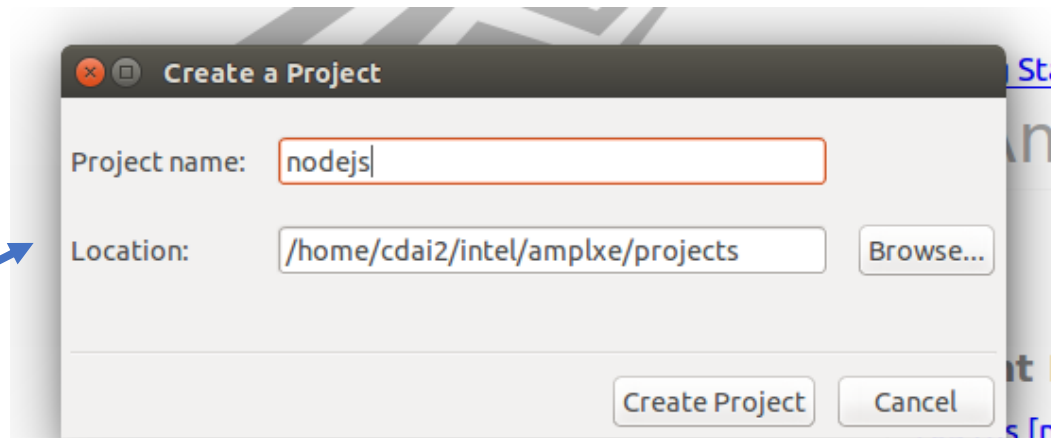
src, build: Enable Intel Vtune profiling for JavaScript.

This feature supports the Intel Vtune profiling support for JITted JavaScript on IA32 / X64 / X32 platform. The advantage of this profiling is that the user / developer of Node.js application can

Profile JavaScript in Vtune (1)

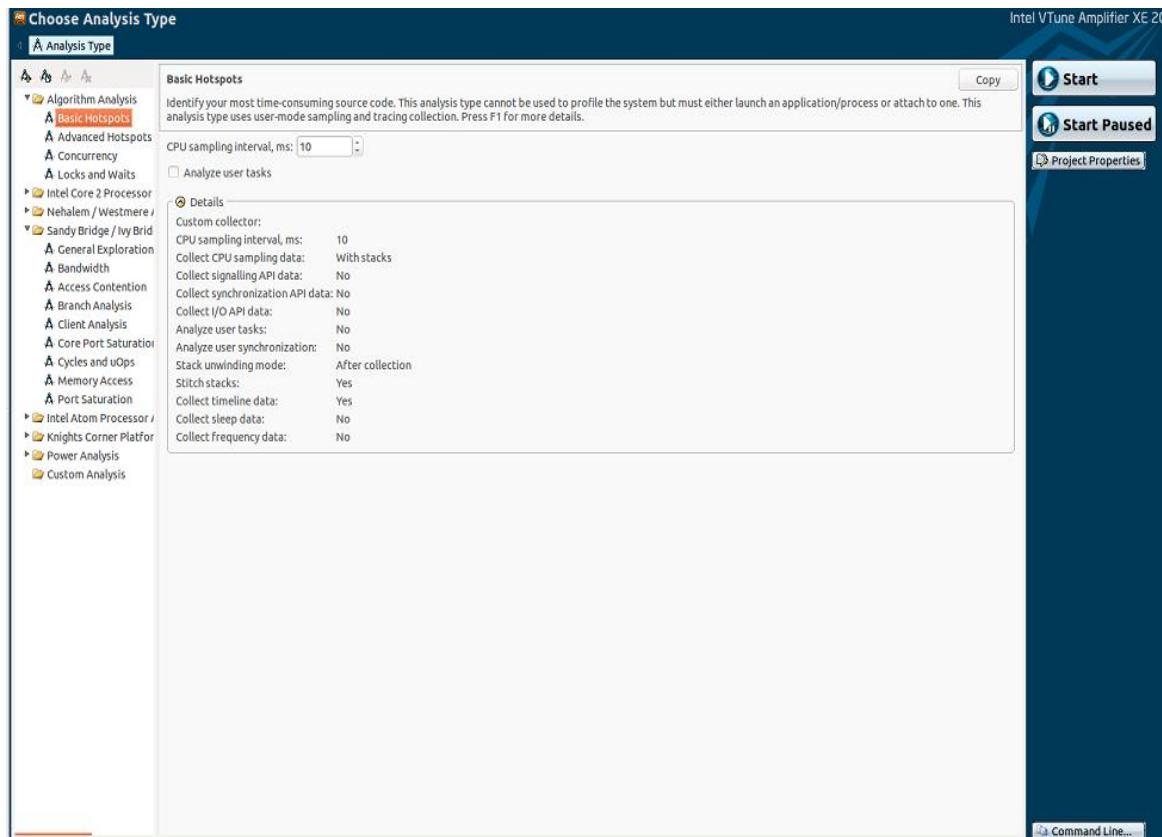
- **How to do Node.js profiling in Vtune:**

- Step 1 : Get/Install [Vtune](#).
- Step 2 : Open Intel VTune and create one project in it.
- Step 3 : Configure VTune project.

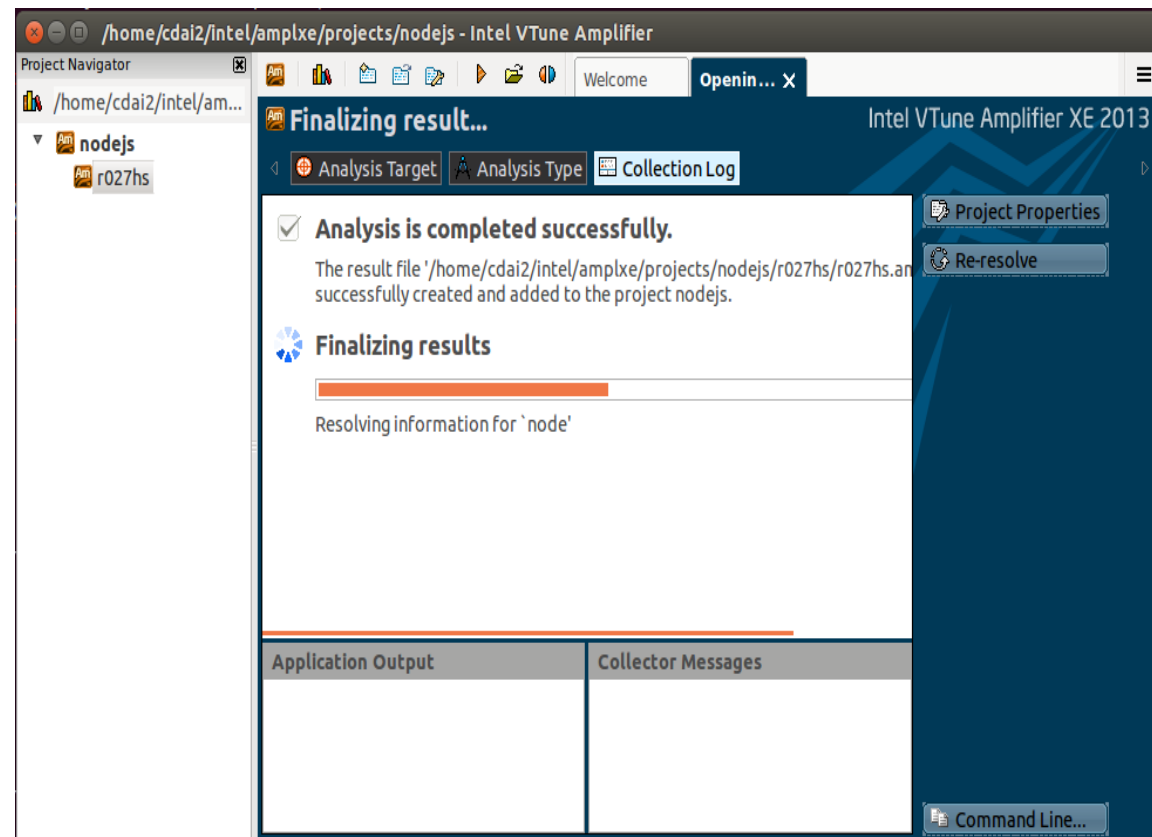


Profile JavaScript in Vtune (2)

- Step 4 : configure profiling type and start profiling



- Step 5: show result do analysis.



Profile JavaScript in Vtune (3)

- Step 5: show result do analysis.

Function/Call Stack	CPU Time by Utilization	Module	Function (Full)	Source File	Start Address
*calcVolatility	6.722s	0s [Dynamic code]	calcVolatility	WebXPRT2013-stockslibrary.js	0x4bf64320
*calcVolatilityAsPercent	6.000s	0s [Dynamic code]	calcVolatilityAsPercent	WebXPRT2013-stockslibrary.js	0x4bf70200
*calcKST	4.270s	0s [Dynamic code]	calcKST	WebXPRT2013-stockslibrary.js	0x4bf70280
*calcSMAday	2.570s	0s [Dynamic code]	calcSMAday	WebXPRT2013-stockslibrary.js	0x4bf63680
*calcSMAofStochOsc	1.678s	0s [Dynamic code]	calcSMAofStochOsc	WebXPRT2013-stockslibrary.js	0x4bf66220
*calcAllTechnicalIndicators	1.502s	0s [Dynamic code]	calcAllTechnicalIndicators	WebXPRT2013-stockslibrary.js	0x4bf745a0
*calcPriceChange	0.948s	0s [Dynamic code]	calcPriceChange	WebXPRT2013-stockslibrary.js	0x4bf67700
*min	0.936s	0s [Dynamic code]	min	native.math.js	0x4bf40840
*max	0.788s	0s [Dynamic code]	max	native.math.js	0x4bf40b40
*abs	0.788s	0s [Dynamic code]	abs	native.math.js	0x4bf39c40
*LoadFastElementStub	0.724s	0s [Dynamic code]	LoadFastElementStub		0x4bf38ca0
*A	0.688s	0s [Dynamic code]	A		0x2851c4e0
*A	0.604s	0s [Dynamic code]	A		0x2851d4e0
*v8:Internal:Runtime_RoundNumber	0.574s	0s node	v8:Internal:Runtime_RoundNumber(int, v8:Internal:...		0x8802910
*calcRSI	0.520s	0s [Dynamic code]	calcRSI	WebXPRT2013-stockslibrary.js	0x4bf39dc0
*apply	0.372s	0s [Dynamic code]	apply		0x2852c120
*A	0.370s	0s [Dynamic code]	A		0x2852e960
*BinaryOpICStub	0.324s	0s [Dynamic code]	BinaryOpICStub		0x28513a80
*calcStochOsc	0.296s	0s [Dynamic code]	calcStochOsc	WebXPRT2013-stockslibrary.js	0x4bf65000
*GrowArrayElementsStub	0.234s	0s [Dynamic code]	GrowArrayElementsStub		0x4bf33fd0
*BinaryOpICStub	0.176s	0s [Dynamic code]	BinaryOpICStub		0x28523a60

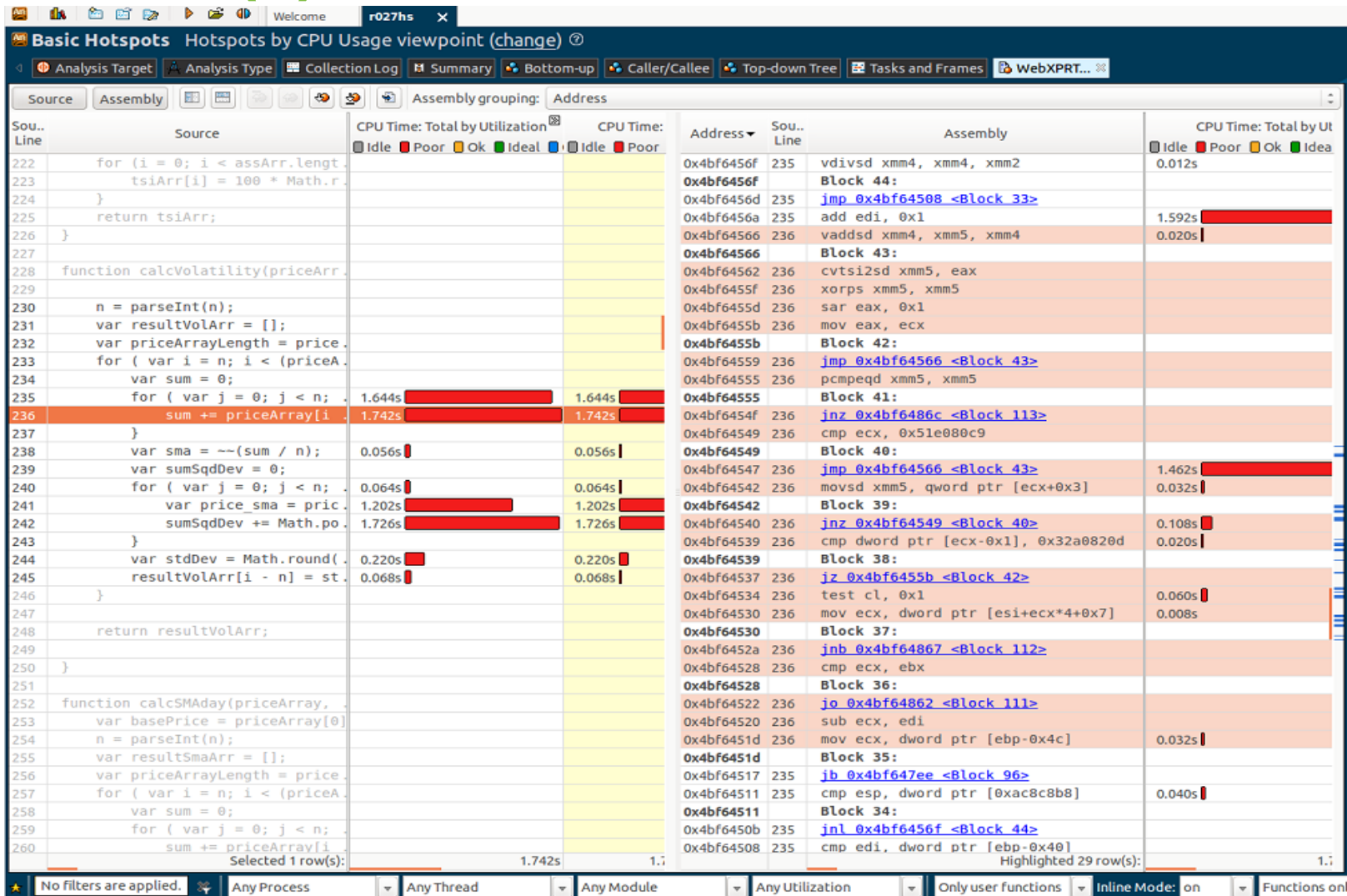
```

222   for ( i = 0; i < assArr.length; i++) {
223     tsiArr[i] = 100 * Math.round((ssiArr[i]
224   )
225   return tsiArr;
226 }
227
228 function calcVolatility(priceArray, n) {
229
230   n = parseInt(n);
231   var resultVolArr = [];
232   var priceArrayLength = priceArray.length;
233   for ( var i = n; i < (priceArrayLength);
234     var sum = 0;
235     for ( var j = 0; j < n; j++) {
236       sum += priceArray[i - j];
237     }
238     var sma = --(sum / n);
239     var sumSqDev = 0;
240     for ( var j = 0; j < n; j++) {
241       var price sma = priceArray[i - j]
242       sumSqDev += Math.pow(price_sma,
243     )
244     var stdDev = Math.round((Math.sqrt(su
245     resultVolArr[i - n] = stdDev;
246   }
247
248   return resultVolArr;
249 }
250 }
251
252 function calcSMAday(priceArray, n) {
253   var basePrice = priceArray[0];
254   n = parseInt(n);
255   var resultSmaArr = [];
256   var priceArrayLength = priceArray.length;
257   for ( var i = n; i < (priceArrayLength);
258     var sum = 0;
259     for ( var i = 0; i < n; i++) {

```

Profile JavaScript in Vtune (4)

- Step 5: show result do analysis.



The screenshot shows the Intel VTune Profiler interface with the following components:

- Top Bar:** Welcome, r027hs, Basic Hotspots, Hotspots by CPU Usage viewpoint (change)
- Navigation:** Analysis Target, Analysis Type, Collection Log, Summary, Bottom-up, Caller/Callee, Top-down Tree, Tasks and Frames, WebXPRT...
- Assembly grouping:** Address
- Source View (Left):** JavaScript code for `calcVolatility` and `calcSMAday` functions. Line 236 is highlighted in orange.
- Assembly View (Right):** Corresponding assembly instructions for the selected line, including `vdivsd`, `jmp`, `add edi`, `vaddsd`, `cvtsi2sd`, `xorps`, `sar`, `mov`, `pcmpq`, `inzb`, `cmp`, `movsd`, `inzb`, `cmp`, `inb`, `sub`, `mov`, `inb`, `sub`, `mov`, `inl`, and `cmp`.
- CPU Time:** Total by Utilization. Legend: Idle (blue), Poor (red), Ok (yellow), Ideal (green).
- Performance Indicators:** Horizontal bars showing utilization for each instruction. Line 236 shows high utilization (red bars).
- Filters:** No filters are applied. Any Process, Any Thread, Any Module, Any Utilization, Only user functions, Inline Mode: on, Functions on.

Agenda

- JavaScript-JITTED x86 machine code mapping profiling support in VTune
 - Background: Scan of current profiling tools for Node.js
 - What's Intel VTune and and the JavaScript code / machine code mapping
 - How to use it in Node.js
- **X87 Quark processor enabling for V8 / Node.js**
 - **Background**
 - **Intel's effort for X87 Quark processor enabling**
 - **How to build Node.js for Quark processor**
- Q / A?

X87 Quark processor enabling for V8 / Node.js

• Background

- **Intel X87:**
 - x87 is only floating point ISA of the x86 architecture instruction set **before** SSE debut
 - x87 co-exist with SSE enable processor
- **Intel Quark Family processor :**
 - Quark processor is designed for ubiquitous computing markets and the Internet of Things (IoT), from automotive to industrial to wearables
 - Quark processor's float point currently is x87-only ISA
- **Google V8's original roadmap of IA32 support :**
 - SSE/none-SSE V8 co-existing in unified IA32/X64 V8 main port before V8 v3.26
 - SSE/none-SSE V8 port is forked since V8 3.27
 - V8 embedded applications such as Node.js needs a dedicated V8 port for X87-only platform after V8 3.26

	<= 3.26 release	> 3.26 release
With SSE2 (IA32/X64)	Yes. With optimized compilers	Yes. With optimized compilers
Non-SSE (including X87-only Quark processor)	Yes. But Low priority. No optimized compiler support.	Not in major x86 master port (Need Intel's work on X87 port)

X87 Quark processor enabling for V8 / Node.js

- **Intel's effort for Node.js enabling on Quark:**
 - **Functionality**
 - Intel created and being maintainer of V8 X87 port inside V8 upstream code repository
 - From V8 v3.27 to current V8 4.8's release branch and future V8 release branch
 - Intel is making sure every release of Node.js works well on Quark since node.js 0.12
 - **Performance**
 - Intel implemented optimized compilers for v8 x87: 1) **Crankshaft** compiler since V8 release v3.27 2) **Turbofan** compiler since V8 release v4.5

X87 Quark processor enabling for V8 / Node.js

- How to build Node.js on Intel Quark processor specifically.

Step 1: download node.js source code.

Step 2: do configuration

```
./configure --dest-cpu=ia32
```

Step 3: modify the config.gypi, add one line as below (marked Red)

```
'target_arch': 'ia32',  
  
.....  
'v8_no_strict_aliasing': 1,  
'v8_optimized_debug': 0,  
'v8_random_seed': 0,  
'v8_use_snapshot': 'true',  
'v8_target_arch': 'x87',  
'want_separate_host_toolset': 1}}
```

Step 4: *./make*

Q / A.

- You can also reach me by email:
chunyang.dai@intel.com