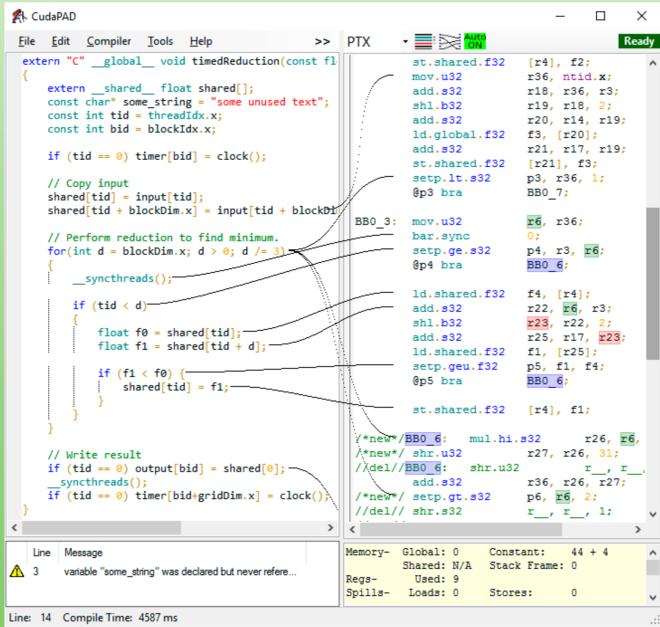


CudaPAD

A Quick On-The-Fly PTX/SASS Viewer For Cuda Kernels.

- Free and Open Source
- Licensed Under the MIT License
- Available on GitHub



CudaPAD is Windows software that aids in the optimization and understanding of Cuda kernels by displaying an on-the-fly view of the PTX/SASS that makes up the kernel. CudaPAD simply shows the PTX/SASS output and does not run any code, however it has several visual aids to help in understanding how minor code tweaks or compiler options can affect the PTX/SASS.

Sometimes when writing Cuda kernels you might wonder what the PTX/SASS might look like and CudaPAD can help with that. It might be for debugging, understanding what's happening, or squeezing in a little bit more performance, or just for curiosity. Just type or paste the kernel in the left panel and then the right panel will display the corresponding disassembly information. Visual aids like visual Cuda-to-PTX code matching lines, WinDiff, and single-click search are built-in to help understand what's happening quickly. Other on-the-fly information is also displayed like register counts, memory usage, compile time, and error information.

With any piece of code, there are often several ways to perform the same task. Sometimes, just changing up a single line or two will lead to different machine instructions with better registers and memory usage. Have fun and make some changes to a kernel and watch how the PTX changes on the right.

— Building CudaPAD —

This project was really fun to create... that is probably why I did so much! Getting the code lines to work was exciting for me. I believe the visual code lines may have been one of the first of their kind when I built this in 2009 but I am not sure. This was a wild idea and I was not sure how complicated it was going to be. Drawing the spline was the easy part but all the miscellaneous stuff like cleaning it up was more difficult. Another difficult part was calculating the location in the text. The textbox line height and line number must be known for each spline drawn. I'm not a graphics expert so I am just happy to get it to work! The visual lines turned out better than expected and are fun to play with.

— Background —

This project was created out of personal need. For some genetic algorithm like algorithms I develop, GPU efficiency is important. One way to help with this is by understanding the low-level mechanics and making any needed adjustments. Before creating this app, I would often get in this loop where I would write a performance critical section of code then view the PTX/SASS over and over using command line tools. Doing this repetitively was time consuming so I decided to automate it with a quick app. It started out as a simple app that would take a kernel in the left window and then output the PTX in a window on the right. This was accomplished by basically running the same command line tools as before but now in an automated fashion in the background. I got carried away and within a short period of time I started adding several features like automatic re-compiling, WinDiff, visual code lines, compile errors, and register/memory usage.

— Using CudaPAD —

In the background, CudaPAD simply compiles the kernels with Cuda tools. The Cuda compiler then in turn calls a C++ compiler like Visual Studio. To run CudaPAD, Cuda and a C++ compiler like Visual Studio needs to be installed.

If the requirements are met, then simply launch executable. When CudaPAD loads, it will have a sample kernel. The sample provides a quick place to start playing around or even a starting framework for a new kernel. Whenever the left-side kernel is edited, it will trigger an update to the PTX or SASS on the right. If there is a compile error, it will show that at the bottom.

System Requirements

- Windows Platform with .NET 4.0
- Visual Studio 2010/2012/2013/2015 (Express editions are okay)
- Nvidia's Cuda 7.0 or 7.5
- A dedicated GPU is not required since nothing is run.

PTX/SASS View Modes

Change the drop down box between PTX, SASS or Source View.

PTX View – shows PTX, the intermediate language, of the kernel. PTX is similar to hardware instructions (SASS) but is a higher level and is less tied to a particular GPU generation.

SASS View – These are true assembly instructions. These types of instructions execute directly on the GPU. The amount of visual information supplied when viewing SASS is less – like the visual code lines do not show.

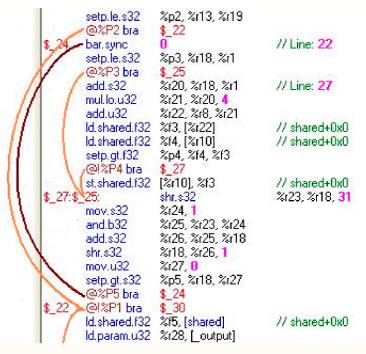
```

/*0148*/ IMAD.U32.U32.HI R0, R0, 0x2, R0; /* 0xa2000000011c0001 */
/*0150*/ SHR R0, R0, 0x1; /* 0xc1480000009c0001 */
/*0158*/ @P1 BRA 0xd0; /* 0x12007fff804003c */
/*0160*/ @P0 BRA.U 0x180; /* 0x120000000c0023c */
/*0168*/ @!P0 LDS R0, [RZ]; /* 0x7a600000023fc02 */
/*0170*/ @!P0 ISCAADD R4, R2, c[0x0][0x144], 0x2; /* 0x60c0080028a00812 */
/*0178*/ @!P0 ST [R4], R0; /* 0xc40000000201000 */
    
```

— Other Ideas —

Below are some ideas to help break down the assembly even further. These features have NOT been added to CudaPAD.

Future Idea #1: Draw curved lines that show jumps. Upward jumps are in a lighter color and downward jumps are in a darker color.



```

mov.u32    %r6, %r38;
bar.sync
setp.ge.s32 %p4, %r3, %r6;
@p4 bra    BB5_6;

ld.shared.f32 %f4, [%r4];
add.s32      %r24, %r6, %r3;
shl.b32     %r25, %r24, 2;
add.s32     %r27, %r17, %r25;
ld.shared.f32 %f1, [%r27];
    
```

Future Idea #2: Click on a register and it would display lines where a register impacts. Dark lines for the actual places the register is used. Gray for registers it impacts after two instructions. This would be similar to Excel's Trace Precedents / Trace Dependents function.

Future Idea #3: Register usage line-by-line. This can display where the greatest register pressure is and where to concentrate on reducing register usage. A /*24*/ would signify that 24 registers are in being used on that line.

```

/*23*/ sub.s32    r18, r3, r38;
/*24*/ shl.b32   r19, r18, 2;
/*24*/ add.s32   r20, r14, r19;
/*24*/ ld.global.f32 f3, [r20];
/*25*/ add.s32   r21, r38, r3;
/*25*/ shl.b32   r22, r21, 2;
/*24*/ add.s32   r17, r17, r22;
    
```

— Advantages of Viewing PTX/SASS —

Trying to figure out that annoying bug. Sometimes viewing the machine instructions can aid in understanding with what exactly is going on.

Curiosity - Sometimes you'll just want to see what is going on under the covers. It is also interesting to see what happens when code is changed. This can be a useful tool for trying to learn about PTX and the compiler.

Changing up a line or two often produces different results. When there exists a kernel that might need some performance optimization, toying with different ways of doing the same thing can produce more efficient code. One example that comes to mind, that has since been fixed, was I found that using a c-style union would result in local memory usage. This is easily spotted in the PTX and I was able to tune that kernel.

```

st.local.s32 [someLocMem], someIntReg; <-- expensive
ld.local.f32 someFloatReg, [someLocMem]; <-- expensive
    
```

Does the code do nothing? Several times in the past, I realized that my kernel had a bug because when I changed or deleted some code nothing changed in the PTX output. I thought to myself, how could this be? One reason why PTX might not change when the kernel is changed is because the compiler often simplifies out useless code that does not do anything. As I found out, this is more common than I expected for my own code as I ran into this a couple of times. This can either be caused by a bug but it could also be just pointless code. In most cases, code that is optimized out should either be removed or fixed. Noticing this can help find some hidden errors in a program.

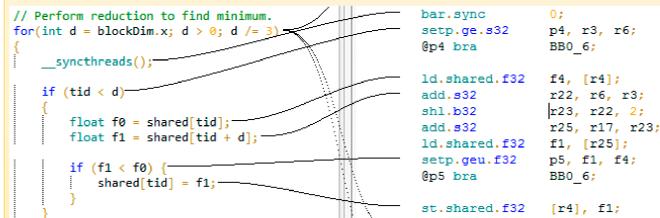
— More Information —

Source Code: <https://github.com/SunsetQuest/CudaPAD>
 Online Article: <http://www.codeproject.com/Articles/999744/CudaPAD>

— Features —

Visual Code Lines

These lines match up Cuda kernel code to the disassembled PTX. They help the programmer quickly identify what code matches up with what PTX. This function can be enabled or disabled.



Syntax Highlighting and Output Formatting

The ScintillaNET textbox control used in this project has some convenient text highlighting abilities that visually helps when viewing code. This results in more colorful and cleaner looking code.

Also, through the use of regular expressions, the text in the output window is cleaned up:

- Removes Unneeded comments
- Remove Empty // comments
- Shorten `__cudaparam__`
- Cleanup Labels
- Remove `.loc` Lines
- More...

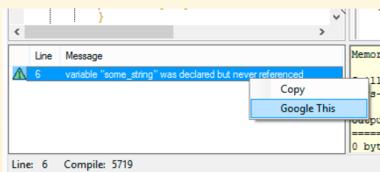
```

.visible_entry timedred(
    .param .u32 timedred_param_0;
    .param .u32 timedred_param_1;
    .param .u32 timedred_param_2;
)
{
    .reg .pred p<0>;
    .reg .f32 f<6>;
    .reg .b32 r<39>;
    ld.param.u32 r8, [timedred_param_0];
    ld.param.u32 r9, [timedred_param_1];
    ld.param.u32 r10, [timedred_param_2];
    cvtb.to.global.u32 r1, r10;
    mov.u32 r2, ctaid.x;
    mov.u32 r3, tid.x;
    setp.ne.s32 p2, r3, 0;
    @p2 bra BB0_2;
    mov.u32 r11, clock;

BB0_2:
    cvtb.to.global.u32 r14, r8;
    /*new*/ shl.b32 r15, r3, 2;
    /*del*/ shl.b32 r_, r_, 2;
    add.s32 r16, r14, r15;
}
    
```

Quick Online Search for Errors/Warnings

Often when running across an error, it is helpful to search for that error online. I found I was often opening a browser and then copying and pasting the error in to a search box. This was not efficient so I added a search online function. At the time, I think this was one of the first of its kind but since 2009, I have seen other IDEs employ this.



Built-In WinDiff

Each time the output window updates, it triggers an automatic a differencing algorithm so the user can see what changed. I decided to add the

```

mov.u32    r38, ntid.x;
/*new*/ sub.s32 r18, r3, r38;
/*del*/ add.s32 r_, r_, r_;
shl.b32    r19, r18, 2;
    
```

WinDiff information inside of a comments in the event the user wants to copy and paste the code. I used //

comments for deleted lines to disable the entire line and the /*new*/ to signify a newly added line.

Tip: To view differencing over several changes, disable the auto compile, make the changes, then click "Start" to force a new re-compile.

Quick Highlighting

Often when working with assembly you want to quickly identify everywhere a register is used. "Find" can be used but that is a bit awkward. In CudaPAD you can just click on a register or any other word in the right window and it will highlight all instances of that register or word. Several items can be highlighted individually, like pictured, with just a few clicks. Click again and it will remove the highlighting as well.

```

shl.b32    r15, r3, 2;
mov.u32    r38, ntid.x;
sub.s32    r18, r3, r38;
shl.b32    r19, r18, 2;
add.s32    r21, r38, r3;
st.shared.f32 [r23], f3;
setp.lt.s32 p3, r38, 1;
    
```

Enabling/Disabling Features

There are several features that can be enabled. All are on by default.

