Programatically detect number of physical processors/cores or if hyper-threading is active on Windows, Mac and Linux

I have a multithreaded c++ application that runs on Windows, Mac and a few Linux flavours.

To make a long story short: In order for it to run at maximum efficiency, I have to be able to instantiate a single thread per physical processor/core. Creating more threads than there are physical processors/cores degrades the performance of my program considerably. I can already correctly detect the number of logical processors/cores correctly on all three of these platforms. To be able to detect the number of physical processors/cores correctly I'll have to detect if hyper-treading is supported AND active.

My question therefore is if there is a way to detect whether hyperthreading is supported AND ENABLED? If so, how exactly.

C/C++/Assembly Programatically detect if hyper-threading is active on Windows, Mac and Linux [duplicate]

This question already has an answer here: Programatically detect number of physical processors/cores or if hyper-threading is active on Windows, Mac and Linux 8 answers I can already correctly

How to detect properly Windows, Linux & Mac operating systems

I could not found anything really efficient to detect correctly what platform (Windows / Linux / Mac) my C# progrma was running on, especially on Mac which returns Unix and can't hardly be differencia

How to detect reliably Mac OS X, iOS, Linux, Windows in C preprocessor?

If here's some cross-platform C/C++ code should be compiled on Mac OS X, iOS, Linux, Windows, how can I detect them reliably during preprocessor process?

JVM + Linux + Intel's Hyperthreading =

I noticed that JVM threads run as processes under Linux for some reasons (correct me if I'm wrong). Furthermore, it is a fact that Intel's Hyperthreading provides additional parallelization only for t

Detect the number of cores on windows

If I am running R on linux or on a mac, I can detect the number of available cores using multicore:::detectCores(). However, there's no windows version of the multicore functions, so I can't use this

How to find physical and logical core number in a kernel module?

Are there kernel functions in Linux that would return the number of the physical core and logical core (in case of Hyperthreading) on which a kernel module is running ?

How can the physical RAM size be determined in Linux programatically?

On the command line this can be found out using the 'free' utility and 'cat /proc/meminfo'. What would be the different ways to find out the physical RAM size in Linux programatically from a : Users

What is crossplatform code snippet for getting totall cores amount in C++? [duplicate]

Possible Duplicate: Programatically detect number of physical processors/cores or if hyper-threading is active on Windows, Mac and Linux I wonder if there is any crossplatform (std or boost based or

How can I query the number of physical cores from MATLAB?

Does anyone know of a way to query the number of physical cores from MATLAB? I would specifically like to get the number of physical rather than logical cores (which can differ when hyperthreading is

How to get screen DPI (linux,mac) programatically?

Hey, I need to know active screen DPI on Linux and Mac OS. I think on linux xlib might be useful, but I can't find a way how to get currect DPI. I want this information to get real screen size in inch

Answers

I don't know that all three expose the information in the same way, but if you can safely assume that the NT kernel will report device information according to the POSIX standard (which NT supposedly has support for), then you could work off that standard.

However, differing of device management is often cited as one of the stumbling blocks to cross platform development. I would at best implement this as three strands of logic, I wouldn't try to write one piece of code to handle all platforms evenly.

Ok, all that's assuming C++. For ASM, I presume you'll only be running on x86 or amd64 CPUs? You'll still need two branch paths, one for each architecture, and you'll need to test Intel separate from AMD (IIRC) but by and large you just check for the CPUID. Is that what you're trying to find? The CPUID from ASM on Intel/AMD family CPUs?

Windows only solution desribed here:

http://msdn.microsoft.com/en-us/library/ms683194

for linux, /proc/cpuinfo file. I am not running linux now so can't give you more detail. You can count physical/logical processor instances. If logical count is twice as physical, then you have HT enabled (true only for x86).

Do you know boost?, assuming you are using C++ I would do it this way, for determining the number of hardware threads available:

```
#include <iostream>
#include <boost/thread.hpp>

int main()
{
    std::cout << boost::thread::hardware_concurrency();
    return 0;
}
```

Note this, does not give the number of physically cores as intended.

EDIT: This is no longer 100% correct due to Intel's ongoing befuddlement.

The way I understand the question is that you are asking how to detect the number of CPU cores vs. CPU threads which is different from detecting the number of logical and physical cores in a system. CPU cores are often not considered physical cores by the OS unless they have their own package or die. So an OS will report that a Core 2 Duo, for example, has 1 physical and 2 logical CPUs and an Intel P4 with hyper-threads will be reported exactly the same way even though 2 hyper-threads vs. 2

CPU cores is a very different thing performance wise.

I struggled with this until I pieced together the solution below, which I believe works for both AMD and Intel processors. As far as I know, and I could be wrong, AMD does not yet have CPU threads but they have provided a way to detect them that I assume will work on future AMD processors which may have CPU threads.

In short here are the steps using the CPUID instruction:

1. **Detect CPU vendor using CPUID function 0**
2. **Check for HTT bit 28 in CPU features EDX from CPUID function 1**
3. **Get the logical core count from EBX[23:16] from CPUID function 1**
4. **Get actual non-threaded CPU core count**
    1. **If vendor == 'GenuineIntel' this is 1 plus EAX[31:26] from CPUID function 4**
    2. **If vendor == 'AuthenticAMD' this is 1 plus ECX[7:0] from CPUID function 0x80000008**

Sounds difficult but here is a, hopefully, platform independent C++ program that does the trick:

```
#include <iostream>
#include <string>

using namespace std;


void cpuID(unsigned i, unsigned regs[4]) {
#ifdef _WIN32
  __cpuid((int *)regs, (int)i);

#else
  asm volatile
```

```
    ("cpuid" : "=a" (regs[0]), "=b" (regs[1]), "=c" (regs[2]), "=d" (regs[3
     : "a" (i), "c" (0));
  // ECX is set to zero for CPUID function 4
#endif
}


int main(int argc, char *argv[]) {
  unsigned regs[4];

  // Get vendor
  char vendor[12];
  cpuID(0, regs);
  ((unsigned *)vendor)[0] = regs[1]; // EBX
  ((unsigned *)vendor)[1] = regs[3]; // EDX
  ((unsigned *)vendor)[2] = regs[2]; // ECX
  string cpuVendor = string(vendor, 12);

  // Get CPU features
  cpuID(1, regs);
  unsigned cpuFeatures = regs[3]; // EDX

  // Logical core count per CPU
  cpuID(1, regs);
  unsigned logical = (regs[1] >> 16) & 0xff; // EBX[23:16]
  cout << " logical cpus: " << logical << endl;
  unsigned cores = logical;

  if (cpuVendor == "GenuineIntel") {
    // Get DCP cache info
    cpuID(4, regs);
    cores = ((regs[0] >> 26) & 0x3f) + 1; // EAX[31:26] + 1

  } else if (cpuVendor == "AuthenticAMD") {
    // Get NC: Number of CPU cores - 1
    cpuID(0x80000008, regs);
    cores = ((unsigned)(regs[2] & 0xff)) + 1; // ECX[7:0] + 1
  }

  cout << "    cpu cores: " << cores << endl;

  // Detect hyper-threads
  bool hyperThreads = cpuFeatures & (1 << 28) && cores < logical;

  cout << "hyper-threads: " << (hyperThreads ? "true" : "false") << endl;
```

```
  return 0;
}
```

I haven't actually tested this on Windows or OSX yet but it should work as the CPUID instruction is valid on i686 machines. Obviously, this wont work for PowerPC but then they don't have hyper-threads either.

Here is the output on a few different Intel machines:

Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz:

```
 logical cpus: 2
    cpu cores: 2
hyper-threads: false
```

Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz:

```
 logical cpus: 4
    cpu cores: 4
hyper-threads: false
```

Intel(R) Xeon(R) CPU E5520 @ 2.27GHz (w/ x2 physical CPU packages):

```
 logical cpus: 16
    cpu cores: 8
hyper-threads: true
```

Intel(R) Pentium(R) 4 CPU 3.00GHz:

```
 logical cpus: 2
    cpu cores: 1
hyper-threads: true
```

OpenMP should do the trick:

```
// test.cpp
#include <omp.h>
#include <iostream>

using namespace std;

int main(int argc, char** argv) {
  int nThreads = omp_get_max_threads();
  cout << "Can run as many as: " << nThreads << " threads." << endl;
}
```

most compilers support OpenMP. If you are using a gcc-based compiler (*nix, MacOS), you need to compile using:

```
$ g++ -fopenmp -o test.o test.cpp
```

(you might also need to tell your compiler to use the stdc++ library):

```
$ g++ -fopenmp -o test.o -lstdc++ test.cpp
```

As far as I know OpenMP was designed to solve this kind of problems.

The current highest voted answer using CPUID appears to be obsolete. It reports both the wrong number of logical and physical processors. This appears to be confirmed from this answer cpuid-on-intel-i7-processors.

Specificly using CPUID.1.EBX[23:16] to get the logical processors or CPUID.4.EAX[31:26]+1 to get the physical ones with Intel processors does not give the correct result on any Intel processor I have.

For Intel CPUID.Bh should be used Intel_thread/Fcore and cache topology. The solution does not appear to be trivial. For AMD a different solution is necessary.

Here is source code by by Intel which reports the correct number of physical and logical cores as well as the correct number of sockets https://software.intel.com/en-us/articles/intel-64-architecture-processor-topology-enumeration/. I tested this on a 80 logical core, 40 physical core, 4 socket Intel system.

Here is source code for AMD http://developer.amd.com/resources/documentation-articles/articles-whitepapers/processor-and-core-enumeration-using-cpuid/. It gave the correct result on my single socket Intel system but not on my four socket system. I don't have a AMD system to test.

I have not dissected the source code yet to find a simple answer (if one exists) with CPUID. It seems that if the solution can change (as it seems to have) that the best solution is to use a library or OS call.

Edit:

Here is a solution for Intel processors with CPUID leaf 11 (Bh). The way to do this is loop over the logical processors and get the x2APIC ID for each logical processor from CPUID and count the number of x2APIC ID's were the least significant bit is zero. For systems without hyper-threading the x2APIC ID will always be even. For systems with hyper-threading each x2APIC ID will have an even and odd version.

```
// input:  eax = functionnumber, ecx = 0
```

```
// output: eax = output[0], ebx = output[1], ecx = output[2], edx = output[
//static inline void cpuid (int output[4], int functionnumber)

int getNumCores(void) {
    //Assuming an Intel processor with CPUID leaf 11
    int cores = 0;
    #pragma omp parallel reduction(+:cores)
    {
        int regs[4];
        cpuid(regs,11);
        if(!(regs[3]&1)) cores++;
    }
    return cores;
}
```

The threads must be bound for this to work. OpenMP by default does not bind threads. Setting export OMP_PROC_BIND=true will bind them or they can be bound in code as shown at thread-affinity-with-windows-msvc-and-openmp.

I tested this on my 4 core/8 HT system and it returned 4 with and without hyper-threading disabled in the BIOS. I also tested in on a 4 socket system with each socket having 10 cores / 20 HT and it returned 40 cores.

AMD processors or older Intel processors without CPUID leaf 11 have to do something different.

On OS X, you can read these values from sysctl(3) (the C API, or the command line utility of the same name). The man page should give you usage information. The following keys may be of interest:

```
$ sysctl hw
hw.ncpu: 24
hw.activecpu: 24
hw.physicalcpu: 12  <-- number of cores
hw.physicalcpu_max: 12
hw.logicalcpu: 24   <-- number of cores including hyper-threaded cores
```

```
hw.logicalcpu_max: 24
hw.packages: 2      <-- number of CPU packages
hw.ncpu = 24
hw.availcpu = 24
```

I know this is an old thread, but no one mentioned hwloc. The hwloc library is available on most Linux distributions and can also be compiled on Windows. The following code will return the number of physical processors. 4 in the case of a i7 CPU.

```
#include <hwloc.h>

int nPhysicalProcessorCount = 0;

hwloc_topology_t sTopology;

if (hwloc_topology_init(&sTopology) == 0 &&
    hwloc_topology_load(sTopology) == 0)
{
    nPhysicalProcessorCount =
        hwloc_get_nbobjs_by_type(sTopology, HWLOC_OBJ_CORE);

    hwloc_topology_destroy(sTopology);
}

if (nPhysicalProcessorCount < 1)
{
#ifdef _OPENMP
    nPhysicalProcessorCount = omp_get_num_procs();
#else
    nPhysicalProcessorCount = 1;
#endif
}
```

From gathering ideas and concepts from some of the above ideas, I have come up with this solution. Please critique.

```
//EDIT INCLUDES

#ifdef _WIN32
```

```
    #include <windows.h>
#elif MACOS
    #include <sys/param.h>
    #include <sys/sysctl.h>
#else
    #include <unistd.h>
#endif
```

For almost every OS, the standard “Get core count” feature returns the logical core count. But in order to get the physical core count, we must first detect if the CPU has hyper threading or not.

```
uint32_t registers[4];
unsigned logicalcpucount;
unsigned physicalcpucount;
#ifdef _WIN32
SYSTEM_INFO systeminfo;
GetSystemInfo( &systeminfo );

logicalcpucount = systeminfo.dwNumberOfProcessors;

#else
logicalcpucount = sysconf( _SC_NPROCESSORS_ONLN );
#endif
```

We now have the logical core count, now in order to get the intended results, we first must check if hyper threading is being used or if it’s even available.

```
__asm__ __volatile__ ("cpuid " :
                      "=a" (registers[0]),
                      "=b" (registers[1]),
                      "=c" (registers[2]),
                      "=d" (registers[3])
                      : "a" (1), "c" (0));

unsigned CPUFeatureSet = registers[3];
bool hyperthreading = CPUFeatureSet & (1 << 28);
```

Because there is not an Intel CPU with hyper threading that will only hyper thread one core (at least not from what I have read). This allows us to find this is a really painless way. If hyper threading is available,the logical processors will be exactly double the physical processors. Otherwise, the operating system will detect a logical processor for every single core. Meaning the logical and the physical core count will be identical.

```
if (hyperthreading){
    physicalcpucount = logicalcpucount / 2;
} else {
    physicalcpucount = logicalcpucount;
}

fprintf (stdout, "LOGICAL: %i/n", logicalcpucount);
fprintf (stdout, "PHYSICAL: %i/n", physicalcpucount);
```

This is very easy to do in Python:

```
$ python -c "import psutil; psutil.cpu_count(logical=False)"
4
```

Maybe you could look at the psutil source code to see what is going on?