

# Cost of calling dynamic libraries

Darryl Gove

*The following text is a section cut from the final version of Darryl Gove's forthcoming book on developing applications on Solaris. The text is copyright Darryl Gove 2007. <http://blogs.sun.com/d/>*

There is a cost associated with calling a dynamic library instead of having a routine linked into the application. The cost is the additional instructions necessary to jump from the application code into the library. This cost is one of the factors that impact the decisions whether to place code in the application, or whether it will be acceptable to place the code in a library. The other significant factors in this is whether the code is going to be shared between many callers, and whether placing the code in a library simplifies the architecture of the application.

The test used to measure the cost of calling a library is to recursively call two libraries, and to time how long it takes to make N library calls. The code for the first library is shown in Figure 1-1. The code for the other library is identical except that the name `jump1` is swapped with the name `jump2`.

```
extern int jump2(int count);

int jump1(int count)
{
    count--;
    if (count==0)
    {
        return 1;
    }
    else
    {
        return 1+jump2(count);
    }
}
```

**Figure 1-1: Code for (jump1.c) one of two recursive libraries**

The two libraries are called from a main routine shown in Figure 1-2.

```
#include <stdio.h>
#include <sys/time.h>

#define RPT 100
#define SIZE 600

static double s_time;

extern int jump1(int count);

void starttime()
{
    s_time=1.0*gethrtime();
}

void endtime(long its)
{
    double e_time=1.0*gethrtime();
    printf("Time per iteration %5.2f ns\n", (e_time-s_time)/(1.0*its));
    s_time=1.0*gethrtime();
}

int main()
{
    int index,count,links,tmp;
    tmp=jump1(100);
    for(links=1; links<100; links++)
    {
        printf("Links = %d ",links);
        starttime();
        for (count=0; count<RPT; count++)
        {
            for (index=0;index<SIZE;index++)
            {
                tmp=jump1(links);
                if (tmp!=links) {printf("mismatch\n");}
            }
        }
        endtime(SIZE*RPT*links);
    }
}
```

**Figure 1-2: Routine to test library call cost**

Figure 1-3 shows the compile lines used to build both the libraries and the static and dynamic test harnesses. In Figure 1-3 the search path for the libraries is specified using the \$ORIGIN token to be in the same directory as the executable. The token needs to be placed in quotes so that the shell does not attempt to interpret it. The application will load both libraries which will satisfy the

```

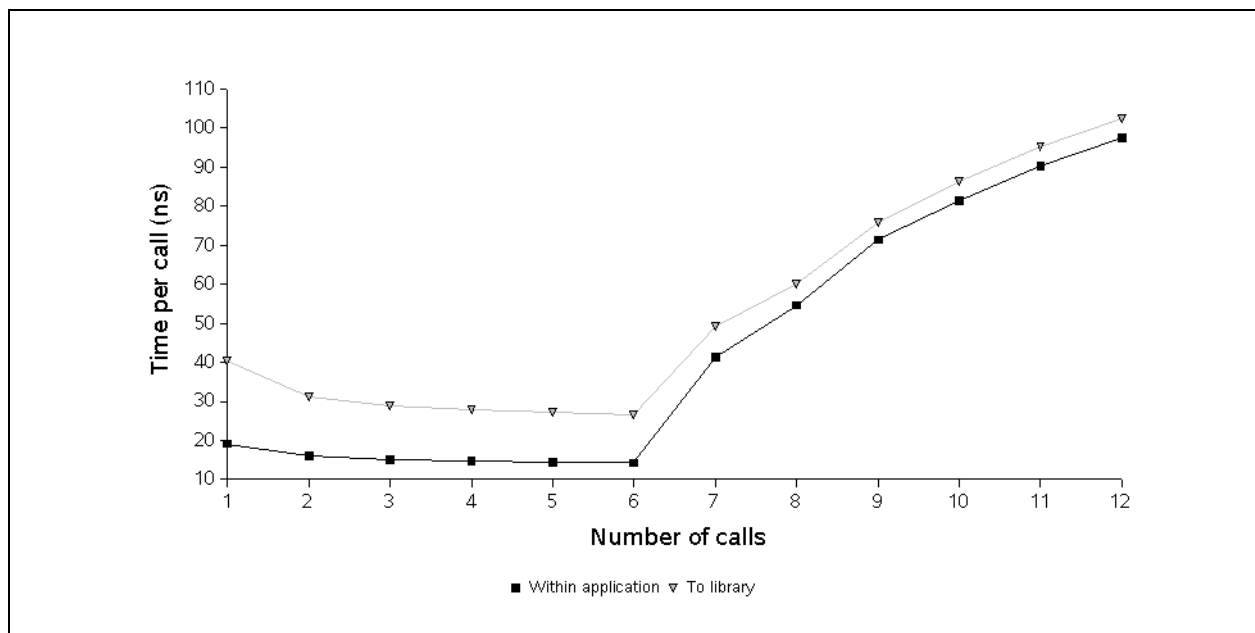
$ cc -O -Kpic -G -o libjump1.so jump1.c
$ cc -O -Kpic -G -o libjump2.so jump2.c
$ cc -O -o dynamic -L. -R'$ORIGIN' -ljump1 -ljump2 main.c
$ cc -O -o static jump1.c jump2.c main.c

```

**Figure 1-3: Building libraries and test harness**

(otherwise circular) links for the two dynamic libraries.

Figure 1-4 shows a graph of the depth of the recursion, and the time in nanoseconds that each call takes.



**Figure 1-4: Cost of calling libraries**

There are a couple of things to observe from this graph.

- First of all the overhead of calling a library rather than calling a function within the application is about 20ns. One way of thinking about this cost is if it represents a large percentage of the time spent actually in the function, then this function would be better placed in the application (particularly if the function is frequently called). On the other hand if the time spent in this function is significantly more than the cost of the call, then putting this function in a library is a good trade-off.
- Secondly, the cost of each of the first six calls remains reasonably constant. However after six calls the cost per call starts increasing. The reason for this is the number of register windows available on the UltraSPARC-III processor. After six calls, the processor has to create a clean register window, and it does this by storing registers to the stack, and later reloading them.