

Re: Translation from recursive to iterative

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2005-03/2633.html

From: Eric Sosman (eric.sosman_at_sun.com)

Date: 03/22/05

Date: Tue, 22 Mar 2005 14:33:44 -0500

BQ wrote:

> *Richard Bos wrote:*

>

>

>>[*BTW, using C++-style comments and tabs in a newsgroup is not a good
>>idea.]*

>

>

> *ehr, sorry.... :-)*

>

>

>>*First thought: get rid of the premature optimisation, get rid of the
>>unnecessary object (yes, I do see the irony):*

>>

>> *void SearchSlaves(unsigned long start_uid, unsigned long end_uid)*

>> {

>> *if (PingSlave(start_uid,end_uid)==VALID_PING_REPLY) {*

>> *if (start_uid == end_uid) {*

>> *AddUidToSlaveList(start_uid);*

>> *return;*

>> }

>> *SearchSlaves(start_uid, (start_uid+end_uid+1)/2) - 1);*

>> *SearchSlaves((start_uid+end_uid+1)/2, end_uid);*

>> }

>> }

>>

>>*Second thought: while this algorithm is efficient if slaves are very
>>sparse,*

>

>

> *-- and this is the case: max.20/30 'slaves', sparse over a range 2¹⁶ wide*

>

>

>>*or relatively sparse and clustered within the range, this is*

>>*probably an efficient algorithm. Every slave is pinged several times,*

>>*though (log₂(end_uid-start_uid) times, circa), so if there are many*

>>*slaves, this is not efficient. Nor is it optimal if there are a*

comp.lang.c: Re: Translation from recursive to iterative

>>reasonable number of slaves peppered through the range. If you suspect
>>one of those may be the case, a straightforward linear search may beat
>>this algorithm.
>
>
> in my situation, a straightforward linear search can't be made.
> Every ping has a 150ms timeout that can't be reduced, and the
> searchspace is very large.
>
>
>>In fact, if PingSlave uses the obvious, naive algorithm for searching
>>its range, this already *is* a linear search – several times over.
>
>
> Slaves answer to the ping if their uid is start_uid<= uid <= end_uid.
> So searching 10/20 slaves requires about 2–3 seconds at most
>
> Anyway, this code is compiled against a microprocessor with very limited
> resources and I have a problem with stack that pushes me toward
> searching an iterative version of this algorithm: I can make at most 32
> subroutine calls, and SearchSlave calls itself at least 16 times in a
> 2¹⁶ space. Since SearchSlave is not at level 0, I sometime obtain a
> stack overflow.
>
> By the way, theory assures that a recursive algorithm can always be
> rewritten in an iterative way. If anyone has an idea...

If you're sure this is the algorithm you want to use,
go ahead and implement it with an explicit stack of your own:
a local array of start/end pairs plus an index indicating
where the stack top is.

```
#include <limits.h>
#define STACK_MAX (sizeof(unsigned long) * CHAR_BIT)
/* ... or perhaps less if you know something about
 * the possible range of inputs.
 */

void SearchSlaves(unsigned long start, unsigned long end) {
    struct { unsigned long start, end; } stack[STACK_MAX];
    int depth;

    /* Begin by pushing the whole range as one pair */
    stack[0].start = start;
    stack[0].end = end;
    depth = 1;

    /* Keep looping as long as the stack still contains
     * unexplored ranges
     */
    while (--depth >= 0) {
```

comp.lang.c: Re: Translation from recursive to iterative

```
/* Pop a range from the stack */
start = stack[depth].start;
end = stack[depth].end;

/* Do the test */
if (PingSlave(start, end) == VALID_PING_REPLY) {
    if (start == end) {
        /* Base case: a trivial range */
        AddUidToSlaveList(start);
    }
    else {
        /* "Recursive" case: split the range and
        * push the two halves
        */
        unsigned long mid = (start + end + 1) / 2;
        stack[depth].start = start;
        stack[depth++].end = mid - 1;
        stack[depth].start = mid;
        stack[depth++].end = end;
    }
}
}
```

This could be cleaned up a bit to eliminate about half of the stack pushes and pops; I've written it this way to show the essentials of the transformation. Also, with some cleverness you might be able to stack just one endpoint of each range.

--
Eric.Sosman@sun.com