Date: Sun, 24 Mar 91 16:47:57 -0500
From: Dave Hanson <drh>
To: preston@rice.edu
Subject: Arena paper in SP&E

> Date: Sun, 24 Mar 91 14:01:42 CST
> From: preston@rice.edu (Preston Briggs)
>
> A friend pointed out your "*Fast Allocation*..." paper in the January 1990 SP&E. It's very nice, both the ideas and the writing, and I expect it will be very useful. However, I think I've discovered a slight performance bug, causing it to use more memory than necessary.
>
> The inline code for allocation is given as
>
> ```
> p = arena[t]->avail;
> if ((arena[t]->avail += k) > arena[t]->limit)
>    p = allocate(k, &arena[t]);
> ```
>
> In the if-condition, you increment the value of "`arena[t]->avail`". This happens even if `allocate` is called. You also increment `avail` at the end of `allocate`. This is normally ok, since `allocate` will be working with a different arena.
>
> But in the final version of `allocate`, which will extend the last arena if possible, an extended arena will have `avail` incremented twice, leaving an unused gap in the current arena.
>
> The simplest correction seems to be a change to the inline code:
>
> ```
> p = arena[t]->avail;
> if (arena[t]->avail + k > arena[t]->limit)
>    p = allocate(k, &arena[t]);
> else
>    arena[t]->avail += k;
> ```

I think your analysis is correct; good detective work!

In practice, I use a different inline allocation macro that can be used in any expression context:

```
#define alloc(n,ap) (ap->avail + (n) > ap->limit ? \
      allocate(n, &ap) : \
      (ap->avail += (n), ap->avail - (n)))
```

e.g., `alloc(k,arena[t])`. As you can see, this macro is equivalent to your version. I wanted to clean it up and avoid using macros for publication, but in doing so, I introduced a bug!