Charm++ - Feature #2042
Add [whenidle] to simplify speculative execution
01/03/2019 10:49 AM - Eric Bohm

<table>
<thead>
<tr>
<th>Status</th>
<th>New</th>
<th>Start date:</th>
<th>01/03/2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Normal</td>
<td>Due date:</td>
<td></td>
</tr>
<tr>
<td>Assignee</td>
<td>Raghavendra Kanakagiri</td>
<td>% Done:</td>
<td>0%</td>
</tr>
<tr>
<td>Category</td>
<td></td>
<td>Estimated time:</td>
<td>0.00 hour</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
<td>Spent time:</td>
<td>0.00 hour</td>
</tr>
</tbody>
</table>

Description
There are various cases where some computation can be done in the absence of other work, but would be of a low priority, or speculative. In such cases it might be desirable to have an entry method triggered by a CcdIdle event which could perform the speculative work if there is nothing better to do. This can currently be manually implemented by setting a Ccd and having the callback queue itself again via CCD, but this involves some moderately ugly code that could be trivially generated automatically. So the only new aspect to this is that it would reduce the burden on the developer to implement this and the activity would be more naturally traced and visible in the Charm++ parallelization instead of hidden in chain of callbacks that don’t even have to be entry methods.

Several refinements could be applied here, possibly as a parameterized elements such as whether the ide callback should only be called once, or every time sustained idle is detected. Furthermore, additional parameterization could set a threshold for how much idleness is necessary to trigger the event.

History
#1 - 01/03/2019 12:21 PM - Sam White
If you have an example of the ugly Ccd code to do this, could you post it here? I anticipate needing to implement this myself soon

#2 - 01/03/2019 12:38 PM - Eric Bohm
Basic pattern (without templates or dependencies on more than C based trickery) is:

```c
static void idleProgress(void *myObj, double time){
    MyClass *m= static_cast<MyClass*>(myObj);
    if(m->speculativeWork()) m->registerIdleProgressFunction();
}

void MyClass::registerIdleProgressFunction()
{
    CcdCallOnCondition(CcdPROCESSOR_STILL_IDLE, &idleProgress, (void *) this);
}
```

In this scheme, the speculativeWork function returns a boolean to indicate that is worth checking for speculative work on future idle invocations. That allows us to avoid future overhead of checking if we know for certain within the scope of the speculativeWork function that there will be nothing more to do. The application can re-register if that changes.