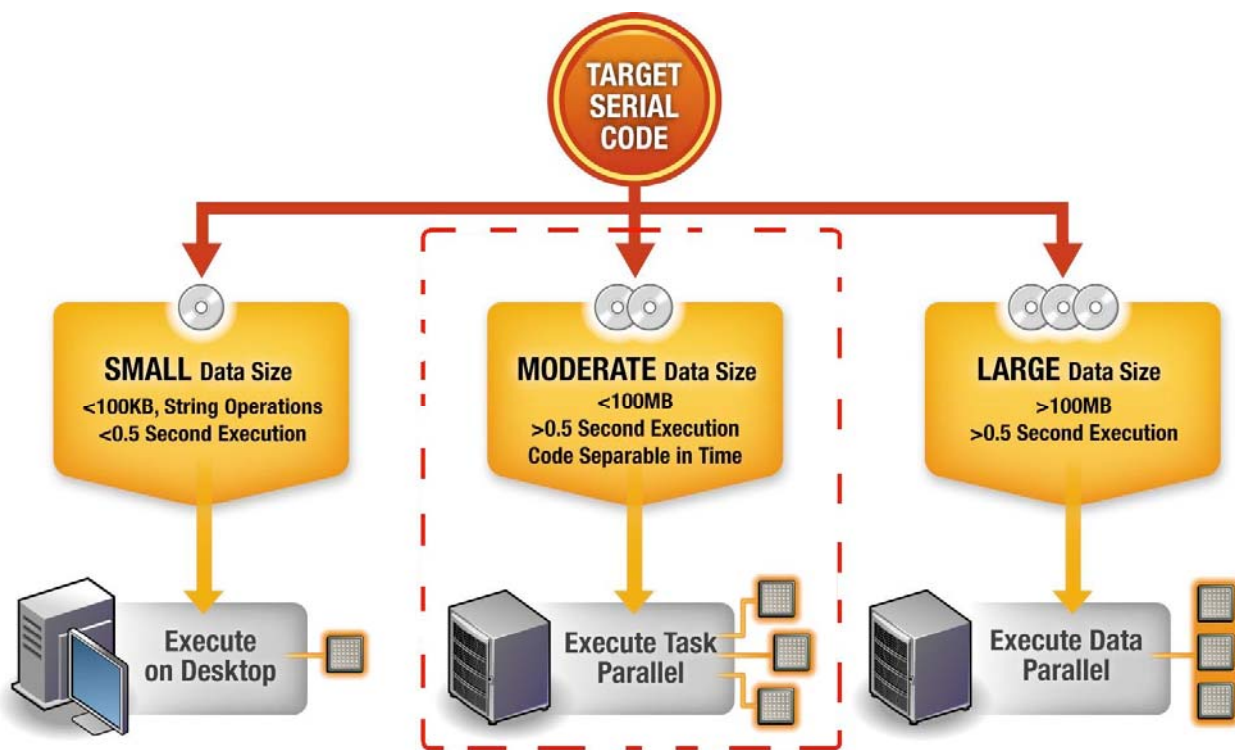


Using Task-Parallel Profiler to Estimate Star-P Speed-Up

The Star-P Task-Parallel profiler provides an estimate of how Star-P might speed up a section of code that lends itself to task-parallel computing, based on the characteristics and code structure. **This estimate will apply only to the sections of your code that are to be computed in a task-parallel (or embarrassingly parallel) fashion.** It does not provide a performance estimate for applications that are inherently data-parallel (also known as fine-grained parallel). Task parallel execution in Star-P needs requires that all the iterations of the for loop that you are targeting with Star-P need to be completely independent from each other. Here, it is up to the user to determine the nature of the parallelism in his/her code and/or identify the specific section of the application that he/she wants to speed up with Star-P.



Running the Task-Parallel Profiler: First download the profiler from the Interactive Supercomputing web site and place it in the same location as the application that you want to profile or put it in a location that is included in you Matlab®¹ path. Then, you can simply run the following command:

```
>> tpprofile <script>
```

¹ STAR-P™ and the “star” logo are trademarks of Interactive Supercomputing. MATLAB® is a registered trademark of The MathWorks, Inc. ISC’s products are not sponsored or endorsed by The MathWorks, Inc. or by any other trademark owner referred to in this document.

where `<script>` is the name of your script. If your application is contained in a function file instead of a script please write a little wrapper script to call your application. After you issue the above command the profiler will run your code five times and produce a report based on its findings. A successful report will look like:

```
TPPROFILE: running testscript first pass...
TPPROFILE: running testscript second pass...
TPPROFILE: running testscript third pass...
TPPROFILE: running testscript fourth pass...
TPPROFILE: running testscript fifth pass...
TPPROFILE: completed. Running report

** Your application ran in 80.2 seconds
   Estimate for your application with Star-P.
   on 32 processors is approximately 5.4 seconds.
```

There are several reasons why a profile run might “fail”:

1. Your application takes less than 30 seconds to run. Applications with very short runtimes are best performed on the desktop to avoid the overhead cost of parallel computation.
2. The profiler detected a code structure that is not optimal for Star-P. In this case you will be presented with a set of numbers that will allow IS C to provide you with advice as to how to restructure your code.
3. The profiler’s estimate is much smaller than the runtime of your application. In this case the profiler is unable to provide a reliable estimate of your application’s runtime with Star-P.

Timing only a subsection of your code: If you know exactly what part of the code in your application corresponds to the task-parallel part that you are interested in speeding up with Star-P, then the profiler supplies a set of markers to delimit the code segment that you want to profile. For example, if you are interested only in the profile for the main FOR loop in your code you can add the `tpprofile start` and `tpprofile end` markers as follows:

```
% Start marker for task-parallel profiler
tpprofile start

% Main for-loop you want to profile:
for i = 1:n
    ...
    ...
end

% End marker for task-parallel profiler
tpprofile end
```

To run the profiler with the markers active, issue the command:

```
>> tpprofile <script> -with-markers
```

This will run the task-parallel profile on your script name `<script>` but only for the marked segment of code.

An example: The example included here is also included as an m-file with the task-parallel profiler on the Interactive Supercomputing website. Say we have a script that implements a simple high frequency filter that needs to be applied to a stack of images. The operations are indeed embarrassingly parallel since each image can be filtered independently from all the other ones. All that is needed is the

mask used in the filter operation.

```
% The example code implements a simple high frequency filter.
%
% Start with creating some fake data: a stack of images that need
% filtering. Also preallocate the output for the filtered images.
npixel = 1000;
nimages = 50;

images = randn(npixel,npixel,nimages);
filtered_images = zeros(size(images));

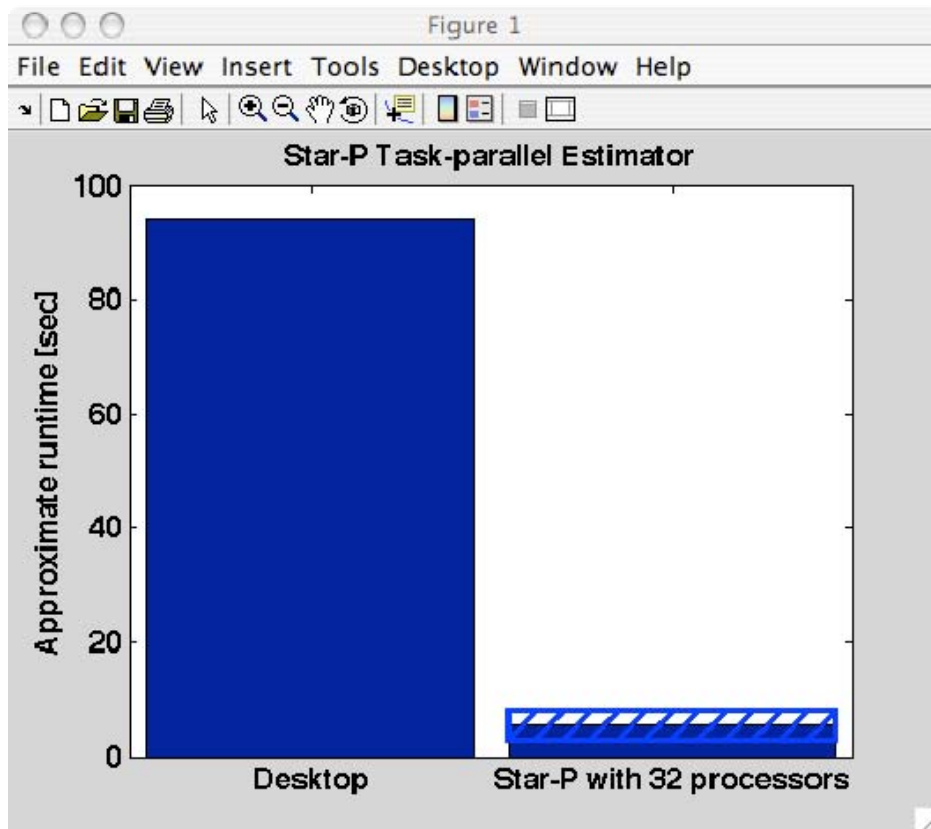
% Define the mask that we will use for as the high frequency filter.
A
% simple Guassian centered on the image center
[maskx, masky] = meshgrid(1:npixel);
width = npixel/ 10;
center = npixel/2;
mask = exp(-0.5*((maskx-center).^2 + (masky-center).^2) / ...
           width.^2) / sqrt(2*pi*width.^2);

% Start profiler here
tpprofile start
%
% Now loop over all images an apply the high frequency filter
for i = 1:nimages
    tmp = fft2(images(:,:,i));
    tmp = fftshift(tmp);
    tmp = tmp .* mask;
    filtered_images(:,:,i) = ifft2(tmp);
end
% End profiler here
tpprofile end
% Done
```

Since we know the section of the code that is task-parallel, i.e., the for-loop at the end of the script, we can use the profile markers to limit the profiling to just that section of the code.

```
>> tpprofile testscript -with-markers
```

This process the report above and the following image displaying the desktop runtime and the esti-



mated Star-P runtime for a system containing 32 processors. The hashed area gives a rough estimate of the uncertainty in the estimate (corresponding to a factor of ~ 2).

Star-P implementation of example script: So now that we profiled our code and determined it would run well with Star-P let's actually implement the Star-P version of this code. To do so we follow the standard protocol to transform a for-loop to a task-parallel call with `ppeval`. First, take the contents of the for-loop and turn it into a function:

```
function filtered_img = mask_func(img)

tmp = fft2(images);
tmp = fftshift(tmp);
tmp = tmp .* mask;
filtered_img = ifft2(tmp);
```

Then, you can call this function with the Star-P command `ppeval`. Note that we also created our data on the Star-P server to increase the parallel efficiency in our code, by adding `*p` constructs to `nimages` and to the argument in the `meshgrid` command.

```

% The example code implements a simple high frequency filter.
%
% Start with creating some fake data: a stack of images that need
% filtering. Also preallocate the output for the filtered images.
npixel = 1000;
nimages = 50*p;

images = randn(npixel,npixel,nimages);
filtered_images = zeros(size(images));

% Define the mask that we will use for as the high frequency filter.
A
% simple Guassian centered on the image center
[maskx, masky] = meshgrid(1:(npixel*p));
width = npixel/ 10;
center = npixel/2;
mask = exp(-0.5*((maskx-center).^2 + (masky-center).^2)/width.^2) /
...
    sqrt(2*pi*width.^2);

%
% Now loop over all images an apply the high frequency filter
filter_images = ppeval('mask_func',images);
% Done

```