FastLap Version: 2.0

FastLap is sub-program which can perform computations required in the analysis of potential problems, such as the solution of integral formulations of Laplace problems and the determination of field quantities due to collections of singularities.

T. Korsmeyer K. Nabors J. White

Research Laboratory of Electronics

Massachusetts Institute of Technology
Cambridge, MA 02139 U.S.A.

October 22, 1996

This work was supported by Advanced Research Projects Agency contracts N00014-87-K-825, MDA972-88-K-008, and N00014-91-J-1698, National Science Foundation contracts MIP-8858764 A02, and ECS-9301189, Office of Naval Research contract N00014-90-J-1085, F.B.I. contract J-FBI-88-067, and grants from I.B.M. Digital Equipment Corporation, and Analog Devices.

Contents

1 Release Notes

1.1 Version 2.0

1.1.1 Major Changes

- The computation of the gradient of the kernel (or Green function) has been coded. This means that the gradient of the field may be computed, and problems cast as the "source formulation" may be solved. For readers not familiar with this jargon, this is the integral equation one obtains by taking the gradient of the single-layer formulation and forming the scalar product with the evaluation point normal vector.

- A generalized preconditioner has been added. Called the `SP` preconditioner, this is an overlapping block preconditioner where the local problems set up for inversion are formed in a more general way than with the `OL` preconditioner. In that preconditioner, local problems are formed by filling a matrix which maps all of the singularities in a cube's first nearest neighbors to all of the evaluation points in the same set of cubes. In the `SP` preconditioner, local problems are formed by filling a matrix which maps all of the singularities in a cube's first nearest neighbors to all of the evaluation points which "belong" with them regardless of whether these evaluation points may be found in this set of cubes. In desingularized problems there is no guarantee that the evaluation point which belongs to any singularity will be found in the same or even a neighboring cube. Therefore the `OL` preconditioning algorithm may result in non-square local problems and poorly organized mappings. *The one-to-one relationship of singularities to evaluation points is deduced from the input.* If no preconditioning is used, there is no need to organized the input to imply this relationship.

1.1.2 Minor Changes

- A bug in `driver.c` has been fixed. There was an incorrect allocation:

  ```
  shape = (int*)calloc(size,sizeof(char));
  ```

  which was changed to:

  ```
  shape = (int*)calloc(size,sizeof(int));
  ```

1.2 Version 1.9

1.2.1 Major Changes

- New options have been added to the capability of `fastlap`. It is now possible to return to the calling program having only made a field computation and

it is possible to specify the right-hand side vector explicitly, as in a single- or double-layer formulation.

- There are now separate data structures for singularities (called `snglrty`, formerly called `charge`) and collocation or field points (or more generally: "evaluation points") (called `fieldpt`). The elements of these structures are now sorted into two (possibly) different spatial decompositions. The singularities are sorted into an oct-tree for the upward pass and the evaluation points are sorted into an oct-tree for the downward pass. This is the primary feature which has allowed for the more general computations now possible with `fastlap`.

- The generalization of the code requires a more complicated interface. The interface is described in detail in §4.2.

1.2.2 Minor Changes

- The convergence of the GMRES procedure is now based on the relative reduction of the L_2 norm of the residual. It used to be absolute. This is a debatable point and a bigger debate could be had about L_∞ versus L_2.

- The Gauss-Jordon inversion has been speeded up a bit by taking an `if` out of the inner loop. Pivoting has been turned off in this routine. It can be turned back on by setting PIVOT to ON in `mulGlobal.h`.

- The orthogonalization in GMRES has been changed from classical Gramm-Schmidt to modified Gramm-Schmidt.

- The direct solution and explicit gmres options have been eliminated. These options were controlled by `EXPGMR` and `DIRSOL`.

- There is no longer a test for redundant panels.

- Influence coefficients for panels are no longer normalized by area. This normalization is not compatible with allowing all sorts of singularities.

1.3 Version 1.0

1.3.1 Major Changes

- Version 1.0 is algorithmically identical to the beta version. Several bugs in the code have been fixed and corrections to the documentation have been made.

- Version 1.0 is a major improvement over the beta version in one respect. This is that this version may be invoked repeatedly by the calling program. This is required by many users who are performing time dependent calculations. The changes to the code all concern memory management. These changes

have no affect on the use of the code in single-call applications. Routines have been changed in the following files: `fastlap.c`, `calcp.c`, `mulLocal.c`, `mulMulti.c`, `uglyalloc.c`.

1.3.2 Minor Changes

- Attempting to solve a system with a null right-hand side is trapped in the procedure `gmres`. The code will abort.
- The iteration loop in the procedure `gmres` has been modified so that at least one iteration will be performed even if the initial trial vector is such that the tolerance criterion is satisfied.
- The distinction between setting `numLev` to zero and unity has been corrected in the documentation.

2 Introduction

This manual describes FastLap (*Fast Lap*lace solver), a procedure for solving general, three-dimensional, Laplace problems on multiply-connected domains via low-order boundary-element (or panel), or de-singularized methods. It is intended that FastLap can replace the typical order N^2 procedures found in many codes requiring the solution of Laplace's equation, and thereby reduce the computational effort and memory requirements of these codes to nearly order N, where N is the number of boundary elements.

Laplace problems in the fields of hydrodynamics, aerodynamics, electro-statics, and others, are often cast as boundary integral equations. Upon discretization, these equations lead to large, dense linear systems which are usually solved by iterative methods. There are two order N^2 tasks associated with these methods: calculating the N^2 entries of the left-hand side matrix when constructing the system of equations, [1] and calculating the iterative method residuals as a dense matrix-vector product. The FastLap algorithm explicitly computes only an order N subset of the matrix elements, and computes the residuals in nearly order N operations. We refer to the technique used by FastLap as a pre-conditioned, adaptive, multipole-accelerated algorithm. For the details of the theory as applied to Laplace problems see [2] and [3], and for more background on: the preconditioner, see [6]; the iterative solver (GMRES), see [4]; and for multipole acceleration, see [7] [8].

The FastLap code itself, is an outgrowth of our work in electro-statics, and is largely a modification and extension of the code FastCap (*Fast Cap*acitance extraction) [1] which solves the first-kind, indirect problems which arise in electrostatics.

This manual describes how to use FastLap for the solution of Laplace problems by a low-order boundary element method. It is assumed that users are familiar with boundary element methods and currently have, or are developing, codes which require the solution of a three dimensional Laplace problem. For most codes of this type, whether in statics or dynamics, the solution of the Laplace problem is the dominant computational burden so that the use of this module rather than a typical order N^2 approach can make a significant difference in the size of problem which can be addressed.

FastLap is written in traditional C, but the procedure is linkable to either C or Fortran calling routines. A `makefile` is provided with the distribution which will make complete example programs with both Fortran and C main programs calling the FastLap procedure. The `makefile` itself serves as an example of how to compile FastLap and link it to your own program, while the example calling programs demonstrate how to solve various different potential problem integral formulations.

[1] When a direct formulation is used, there will be a similar order N^2 effort for the right-hand side.

3 FastLap Overview

FastLap has been created to accelerate the solution of Laplace problems cast as boundary integral equations. The specific computation which is accelerated by the `fastlap` procedure is the evaluation of the potential at M field or collocation points (or more generally: "evaluation points") due to N singularities. That is

$$b = Cq \tag{1}$$

in which the matrix $C \in \Re^{M \times N}$ contains the influence coefficients for the N singularities, whatever their type, the vector $q \in \Re^N$ contains the strengths of these singularities, and the vector $b \in \Re^M$ is the potential (or its gradient) at the M evaluation points. The important concept to understand is that this computation of a potential field appears in the inner loop of a typical iterative technique used to solve a boundary integral equation. As such, the potential field evaluation may be required many times (although much less than N times, one hopes) to acheive convergence of the iterative solution and may be required upon solution to evaluate quantities on and away from the solution surface.

As a matter of convenience, FastLap incorporates a preconditioned GMRES solution algorithm for linear systems. Using this feature imposes some restrictions on the problems which may be solved. FastLap Version 2.0 supports:

- square linear systems (*i.e.* $M = N$);
- constant distributions of sources and dipoles on quadralateral and triangular panels, points sources and dipoles, and gradients of these;[2]
- Green's Theorem, single-layer, source formulation, and desingularized formulation.

An alternative to using the preconditioned GMRES solution algorithm for linear systems is to request simply a field computation (1). In this case, the iterative solution procedure is taking place in the calling routine and b may not be the exactly the vector required in that solution. In other words, if a large part of the problem may be represented with the types of singularities supported by FastLap, then this portion may be accelerated while some smaller portion is handled in a more traditional way. As an example to which this idea would be applied, consider an airfoil problem in which the trailing edge and initial wake panels are of higher order than the body and wake panels which have contant strength. As long as only an order N portion of the

[2]Inclusion of other singularities requires the coding of the near-field (usually analytic) algorithm and the far-field (spherical harmonic expansion) algorithm into the if-blocks that select the algorithms to use depending on the singularity type. Singularity influences which require more than one element of q to evaluate (such as some higher-order panel representations) would be difficult to include. Others, such as vortex blobs are relatively straightforward.

problem is treated directly, great computational savings may be acheived by using FastLap with a little imagination.

Allowing `fastlap` to be a flexible procedure has resulted in a somewhat complicated interface. Early versions of FastLap assumed constant strength panels with collocation at the panel centroids and so the input consisted mainly of a collection of panel vertex coordinates. Added flexibility requires that both the singularities and the evaluation points be specified along with what type of calculation should be associated with each. Take the case of the general Green formulation discretized with constant-strength, planar panels and collocated at panel centroids The calling program will have to provide a pointer to an array of panel vertices, a pointer to a list of singularity types (*i.e.* `CONSTANT SOURCE` or `CONSTANT DIPOLE`) for the inclusion of these singularities on the right- and left-hand sides, a pointer to a vector of boundary data for these panels for the right-hand side computation, and finally a pointer to an array of collocation points (these might be the panel centroids in this case, but need not be and in any case need not have a global ordering consistent to the singularity global ordering). Compare this with Version 1.0, where the input is a pointer to the panel vertex array, a pointer to the panel type, and a pointer to the boundary data. In §s:inter the `fastlap` interface is explained and examples of its use are found in the sample drivers.

The key internal feature of FastLap that allows for flexibility, is the creation of separate data structures and cube hierarchies for the singularities and the evaluation points. The singularity cube hierarchy is associated with the upward pass or the accumulation of multipole expansions, and the evaluation point hierarchy is associated with the downward pass or the distribution of local expansions. (For a description of these upward and downward passes see the algorithm description in [2]). That is, the upward pass is concerned with accumulating the contributions of the singularities to the potential, so it takes place in a cube hierarchy determined by the sorting of the singularities. On the other hand, the downward pass is concerned with distributing the contributions to the evaluation points, so it takes place in a cube hierarchy determined by the sorting of the evaluation points. There is no reason for these hierarchies to have any part of their structure in common aside from the root cube. It is the computation of the interaction field (the conversion of multipole to local expansions) and the direct field (the computation of the influence of a specific singularity at a specific evaluation point), which links the two hierarchies.

4 Combining FastLap With Your Code

FastLap is a linkable procedure which you can integrate into your own code to accelerate the computations required in the analyses of potential theory. This section describes how to perform this program integration and how to configure the FastLap module to your particular requirements. Example drivers, in both C and Fortran, are provided to guide you in the use of FastLap . Making and using these drivers are described in §7. How to unpack the distribution is described in §8.

4.1 Compiling FastLap and Linking to C or Fortran

This section describes the procedure to create an executable code which calls the FastLap procedure. Normally, FastLap does not produce any output beyond warnings and fatal errors. However, you may wish to run FastLap in a diagnostic mode where cpu time and memory use as well as other details are reported. In that case, certain flags should be set before compilation. For a description of these flags, see §6.1

The FastLap module is written in traditional C, but it is linkable to both C and Fortran. As provided in the distribution, the module is configured for linking to C. For linking to Fortran on many systems, the name of the top level procedure, `fastlap`, must be changed to `fastlap_`. There is a command in the `makefile` provided with the distribution which will perform this modification for you when you create the example executable which has a main program written in Fortran. If your compiler does not require this, use the alternative line provided in the makefile in the rules section for `fastlapf.c`. Also note that for mixed language programs the linker is invoked with the generic compile/link command of the language of the main program. Use this `makefile` as a guide when you create or modify the `makefile` for your own code, whether the main program is written in C or Fortran.

You may wish to change the compile or link flags in the `makefile` to conform to your particular needs or to the peculiarities of your operating system or compiler. For instance you may wish to set a certain level of optimization. We find that with many compilers using the highest level of optimization possible produces executables which are nearly twice as fast as executables compiled with no optimization.

In the case where you wish to collect and display FastLap execution timing information, *i.e.* the `TIMDAT` flag (see Table 6.1) is set to `ON`, you will *have* to modify the compile flags in the makefile to suit your operating system. Alternative compilation procedures for popular operating systems have been provided for you as choices for the definition of the variable `CFLAGS`. Select the one which corresponds to your operating system and leave the others commented out.

AIX system users should use the AIX configuration regardless of whether the timing routines are activated.

IRIX system users should note that their C compiler may default to ANSI C rather than traditional C, and so should use the `-cckr` flag.

HP-UX system users should note that we have had difficulties with memory management under HP-UX C. Use the `DEFINE`'s found in uglyalloc.c to implement the C library routines malloc and calloc rather than those we provide to get started, but the difficulties may not end there if `fastlap` is called repeatedly as in a transient problem.

4.2 The Interface to FastLap

`fastlap` is a flexible procedure which allows a number of different problem solutions and computations. Consequently, setting up the input to `fastlap` properly is not trivial. While the interface is described here, it is very useful to look carefully at the code of the example drivers which are provided with the distribution. Most of the possible uses of `fastlap` are demonstrated in those codes.

Recall that in Fortran, arguments are passed to procedures by address only, while in C, they are passed by value (which could be an address). For compatibility with both C and Fortran, `fastlap` expects all arguments to be addresses.

Multi-dimension arrays are a problem when a Fortran program calls a C procedure, as the C procedure has no information on the configuration of the array axes in memory. The work-around is that any argument which might logically be set up as a multi-dimension array must be set up with a single axis (as a vector). This is the case for the coordinates of the panel vertices, and so the example programs described in §7 are particularly helpful on this point.

The interface to FastLap looks like:

```
int fastlap(plhsSize, prhsSize, pnumSing, px, pshape, pdtype,
   plhsType, prhsType, plhsIndex, prhsIndex, plhsVect, prhsVect,
   pxf, pxnrm, pnumLev, pnumMom, pmaxItr, ptol, pjob)

int *plhsSize, *prhsSize, *pnumSing, *pshape, *pdtype,
   *plhsType, *prhsType, *plhsIndex, *prhsIndex, *pnumLev,
   *pnumMom, *pmaxItr, *pjob;

double *px, *plhsVect, *prhsVect, *pxf, *pxnrm, *ptol;
```

In which:

`fastlap` returns an integer. This integer is the number of iterations performed by the GMRES procedure. If it is equal to `maxItr`, then the GMRES procedure may not have converged and `tol` should be examined. If it is equal to zero, then no GMRES iterations have taken place (this is a correct return if a field computation has been performed (see `job` below)).

`plhsSize` points to an integer specifying the number of unknown singularity strengths on the left-hand side and the number of evaluation points. (*N.B.* solution of under- or over-determined systems is not supported.) This integer is represented by N in this text.

`plhsSize` points to an integer specifying the number of known singularity strengths on the right-hand side. This integer need not equal the number of singularity strengths on the left-hand side, or the number of evaluation points.

`pnumSing` points to an integer specifying the total number of singularities in the problem whether on the right- or left-hand side. This integer is represented by M in this text.

`px` points to a double precision vector of length $3*4*M$. px[(i*4*3)+(j*3)+k] is the coordinate x_k of the j^{th} node of the i^{th} singularity. If the i^{th} singularity is a quadrilateral panel, then there will be twelve such numbers; if the i^{th} singularity is a triangular panel, then there will be nine; and if the i^{th} singularity is a point singularity, then there will be three. The ordering of the vertices for panels $k = 1, 2, 3, (4)$ must be such that they appear clockwise to an observer *inside* the computational domain (this insures the correct sign on the self influence of a dipole distribution).

`pshape` points to an integer vector of length M. `pshape[i]` indicates the number of vertices which make up the geometry of the singularity. The currently supported possibilities are 1 for point singularities, 3 for triangular panels, and 4 for quadrilateral panels.

`pdtype` points to an integer vector of length M. `pdtype[i]` indicates whether the source and/or dipole are to be evaluated for the i^{th} singularity, in which case `pdtype[i] = 0`; or the scalar product of the vector `pxnrm` with the gradient of the source and/or dipole are to be evaluated for the i^{th} singularity, in which case `pdtype[i] = 1`. The specification of `pxnrm` allows several different computations to be made, see below.

`plhsType`, and `prhsType` point to integer vectors of length M. `prhsType[i]` or `plhsType[i]` indicates the type of the i^{th} singularity when its influence is accounted for on the right- or left-hand sides of the equation. In the typical Green formulation discretized by a panel method, the i^{th} singularity is a panel and appears as a dipole distribution on the left-hand side and a source distribution on the right-hand side or *vice versa*. `prhsType[i] or plhsType[i]` may be `NULL`, in which case the panel does not appear in the calculation on the corresponding side of the equation.

`plhsIndex`, `prhsIndex` point to integer vectors of length $n*M$, where n is a small, positive integer. n corresponds to the order of the approximation of the solution

and boundary conditions. For instance if the problem were discretized by linear-strength, planar triangular panels, then 3 strengths would be required for each panel and $n = 3$. Currently, only $n = 1$ is supported, so these index vectors are of length M. So `prhsVect[prhsIndex[i]]` or `plhsVect[plhsIndex[i]]` provides the strength for the i^{th} singularity when its influence is evaluated on the right- and left-hand sides of the equation, respectively. Note that the length of the vectors being pointed to is M. If the i^{th} panel is `NULL` on one side of the equation it does not make any difference what element of `prhsVect` or `plhsVect` is pointed to, or `prhsIndex[i]` or `plhsIndex[i]` may be set to zero.

`plhsVect, prhsVect` point to double precision vectors. The length of these vectors is determined by the problem being solved. `plhsVect` points to a vector of length N. `prhsVect` points to a vector which must be no greater in length than $n * M$. `prhsVect[i]` contains the value of the i^{th} boundary condition or the known strength of the i^{th} singularity. On return, `plhsVect[i]` contains the value of the solution at the i^{th} evaluation point if a linear system has been solved, (see `job` below), or `plhsVect[i]` contains the value of the potential at the i^{th} evaluation point if a field computation has been performed (see `job` below).

`pxf` points to a double precision vector of length $3*N$. `xf[3*i+j]` is the j^{th} coordinate of the i^{th} evaluation point.

`pxnrm` points to a double precision vector of length $3 * N$. `xf[3*i+j]` is the j^{th} component of a vector associated with the i^{th} evaluation point. This vector is used when `pdtype[i] = 1`. If the vector provided is the surface normal at the i^{th} evaluation point, the source formulation may be solved. If the vector provided is, for instance, $(1, 0, 0)$, the derivative with respect to x_1 of the field may be calculated.

`pnumLev` points to an integer specifying the depth to which the computational domain will be hierarchically decomposed. If `*pnumLev` is set to one, there will be no multipole acceleration. [In Version 1.0, if `*pnumLev` were set less than one or greater than the maximum number of levels allowed (`MAXDEP` in `mulGlobal.h`) then the depth would be chosen automatically. *This feature is not supported in Version 2.0.*]

`pnumMom` points to an integer specifying the order of the multipole expansions.

`pmaxItr` points to an integer specifying the maximum number of iterations allowed for the GMRES algorithm. The GMRES procedure will cease iterating when the norm of the residual is less than `*ptol` or after `*pmaxItr` iterations.

`ptol` points to a double precision number. On entry `*ptol` specifies the convergence tolerance on the norm of the residual in the GMRES algorithm. On exit `*ptol`

is equal to the norm of the residual at the completion of the GMRES iterations. The GMRES procedure will cease iterating when the norm of the residual is less than `*ptol` or after `*pmaxItr` iterations.

`pjob` points to an integer. If `*pjob = 0` then `fastlap` returns in the vector pointed to by `plhsVect` the field at `*plhsSize` evaluation points located at the coordinates pointed to by `pxf` due to the `*pnumSing` singularities located at the coordinates pointed to by `px` with strengths pointed to by `prhsVect`. If `*pjob = 1` or `*pjob = 2` then `fastlap` returns in the vector pointed to by `plhsVect` the solution vector at `*plhsSize` evaluation points located at the coordinates pointed to by `pxf` due to the `*pnumSing` singularities located at the coordinates pointed to by `px`. If `*pjob = 1` then `*prhsSize` of these singularities appear on the right-hand side with strengths corresponding to the boundary conditions pointed to by `prhsVect` (a Green's theorem formulation). If `*pjob = 2` then the right-hand side is simply a known vector of length `*prhsSize` with elements pointed to by `prhsVect` (an indirect formulation).

5 Strategies for Effective Use of FastLap

The parameters: `size`, `numMom`, `numLev`, and `tol`. all affect the cpu time and memory required for a solution. The effect of `size` and `numMom` on the cpu time required for a solution is approximately:

$$\text{Time} \sim \text{size} * \text{numMom}^4$$

and

$$\text{Memory} \sim \text{size} * \text{numMom}^2$$

The effect of `numLev` is difficult to predict because it depends on the condition of the linear system. As `numLev` increases, the result is that more of the matrix-vector product is performed by the multipole approximation, so increasing `numLev` may increase the efficiency of the computation. However, as `numLev` increases, the preconditioner will have smaller overlapping sub-matrices to invert because the finest-grain nearest neighbors will include a decreasing number of panels, so increasing `numLev` may increase the number of iterations required to pass the tolerance test.

The effect of `tol` is also difficult to predict because it is coupled with `size`. In practical computations, the set-up of the data structures may account for 50% of the computation time, so increasing `tol` and causing several more matrix-vector products to be performed may only increase the total time required by 10 to 20%.
The following are some suggestions on how to pick these parameters:

`*pnumSing` is the number of singularities used to characterize the problem. The calling routine is responsible for the discretization and FastLap does not alter this discretization in any way. Note that with this accelerated algorithm one is free to use finer discretizations and attack larger problems than one is normally accustomed to.

`*pnumMom` controls the accuracy with which singularity influences are computed when they are approximated by multipole expansions. Since a portion of the computing time increases like the fourth power of the number of multipole coefficients, it is important to make this parameter as small as possible. Unfortunately, the error bounds in [7] are much too conservative to provide guidance and practical bounds are not yet available. It is suggested that one begin with a low order, such as 2, and then increase the order to check the accuracy.

`*pnumLev` controls the depth to which the computational domain is hierarchically decomposed. In problems for which preconditioning is supported and has been selected, the choice of `*pnumLev` can greatly alter the computing time for a problem. This is because as `*pnumLev` is reduced, more of the problem is done directly, which increases the computation times, but it is the direct problem

which is the basis of the preconditioning, so the convergence of the GMRES procedure may be accelerated. Experimentation with your particular type of problem is warranted. In many problems it is advantageous to select `*pnumLev` so that there are only a few singularities in a finest grain cube.

If `numLev` is set to one, then all panels are nearest neighbors. In this case, all interactions will be computed directly with order N^2 effort and with order N^2 memory required.

`*ptol` controls the accuracy of the solution of the linear system regardless of the choice of the accuracy of the representation of the interactions of the panels. It is important to note that Krylov subspace methods like GMRES have effort and storage requirements which increase with the square of the number of iterations, so the overall effort and storage requirements for computing the iterates is roughly order N *only as long as the number of iterations is much less than* N. [3]

[3] In our experience, the preconditioners ensure this.

Flag		Function
NOWARN	OFF	Prints non-fatal warnings.
CMDDAT	ON	Prints brief FastLap configuration information.
ITRDAT	ON	Prints the residual norm after each iteration.
TIMDAT	ON	Prints a summary of CPU time and memory usage. Times are only reported if FastLap is compiled using the correct flags for your operating system. See §4.1
CFGDAT	ON	Prints core configuration flags.
MULDAT	ON	Prints brief multipole setup information.
DISSYN	ON	Prints summary of cube involvement by partitioning level.
DMTCNT	ON	Prints the number of multipole transformation matrices for all possible cube pairings.

Table 1: FastLap output configuration compile flags (defined in mulGlobal.h).

6 Messages: Diagnostics, Warnings, and Errors

There are three types of output produced by FastLap . Diagnostic output which can inform you about how the solution is being computed is written to stdout if you request it. Warnings about fragile aspects of your solution are also written to stdout unless you suppress them. Finally fatal errors are written to stderr before execution is halted.

6.1 Setting Output Flags

FastLap provides diagnostic output which is turned on by setting flags in mulGlobal.h under the heading:

```
/* Output Format Configuration */
```

This section describes what output is available, and how the flags can be set to obtain such output.

If any of the global flags in 6.1 are set for printing, then the associated information will be written to stdout. This information includes non-fatal warnings, resource usage, and the algorithm configuration. If all flags are all set to OFF then the only information written will be non-fatal warnings. Regardless of flag settings, fatal error messages are written to stderr.

Table 6.1 describes the output configuration flags in more detail than is found in the comments in mulGlobal.h.

6.2 Warnings

These warnings will appear on stdout if `NOWARN` is set to `OFF` (see Table 6.1). The first part of every warning contains the name of the procedure from which it was written. The following is a list of warnings which may be written by FastLap and explanations which should help you in diagnosing problems. Values which appear in quotes (" ") are for example only.

`FLW-initcalcp: Panel skewed beyond the PLANAR TOL tolerance.` The four vertices of the panel are out of the plane passing through the four midpoints of the four straight sides which connect the four vertices by an amount beyond the tolerance set by the variable `PLANAR TOL`. This quantity is defined in the file `calcp.c` by the line: `define PLANAR TOL 1.0e-3`. Approximating skewed quadrilaterals by planar quadrilaterals contributes to discretization error. This is not an exclusive feature of the multipole accelerated algorithm but rather of planar panel algorithms in general. You should set `PLANAR TOL` to whatever value your experience leads you to have confidence in.

`FLW-gmres: exiting without converging.` The GMRES iterative system solver has failed to converge to within `tol` after `maxItr` iterations. This is not fatal, as your calling program may wish to supply a remedy and continue.

`FLW-mulMatUp: no multipole expansions at level ''5'' (lowest)`

`FLW-mulMatUp: no multipole expansions at level ''4''`

`FLW-mulMatUp: no multipole acceleration.` At any level, FastLap always checks to see whether multipole acceleration is warranted. At finest grain (lowest) there will be no acceleration, and with smaller problems this will continue to be the case at levels closer to level zero. If no level will benefit from multipole acceleration, then the last message will be written.

`FLW-fastlap: compilation with OTHER flag gives incorrect times.` The correct compilation flags are required if `TIMDAT` is set to `ON`. See §4.1.

`FLW-placeq: oversized panel, cube length = 0.01 panel length = 0.02`
If a very non-uniform surface discretization is used, panels may be larger than a finest-grain cube. This violates the conditions upon which the theorems for the error bounds are founded [7]. These error bounds are rather conservative in practice and so panels which are only slightly larger than a finest-grain cube (less than approximately 20% larger) are probably acceptable, but this condition may result in a loss of accuracy. The remedy is to provide a more uniform discretization or to reduce the number of decomposition levels.

6.3 Errors

The messages reported to `stderr` are fatal and when they appear FastLap will terminate. The first part of every error message contains the name of the procedure from which it was written.

6.3.1 Input Errors

FastLap screens the input for unacceptable data and reports such data if it is detected. Other input related errors are detected during a run, particularly when memory is allocated. The following is a list of error messages which may be written by FastLap and explanations which should help you in diagnosing problems:

`FLE-initcalcp: Convex panel in input.` There is a quadrilateral panel for which after projection onto the plane (see the description of `x` in the §4.2) one vertex is in the interior of the triangle which is formed by line segments connecting the other three vertices. Such panels are illegal.

`FLE-fastlap: bad expansion order: ''8''` The value of `numMom` is less than zero, or greater than `MAXORDER`. `MAXORDER` is defined in `mulGlobal.h`.

`FLE-fastlap: Missing boundary condition.` The input data list for a panel is not complete.

```
FLE-./mulSetup.c: out of memory at line ''158''

(NULL pointer on ''1024'' byte request)
```

`FLE-placeq: ''7'' levels set up` Your problem requires more virtual address space than is available. The last line indicates that seven levels have been set up properly, so if you set `numLev` to seven, you should be able to complete the run. You may run out of memory in other sections of the code besides where the multipole matrices are being set up. In that case you should reduce `numLev` to one less than the level reported under `MULTIPOLE SETUP SUMMARY` (which is reported if `MULDAT` is set to `ON`, See Table 6.1).

6.3.2 Run-Time Errors

¡FastLap is a program under development and as such is likely to have a variety of bugs. There are quite a few additional error messages which you may encounter *en route* to finding these bugs. It is hoped that these messages will be helpful.

7 Example Drivers for FastLap

The content of this distribution is based on the assumption that *fastlap* users are sophisticated numerical analysts. Therefore we think that examples are more useful than detailed explanations. In that spirit there are several example programs provided in the distribution, written in C and Fortran, which invoke FastLap for the solution of a various different problem formulations. To see what is available, type `make`. These example programs obtain solution parameters from the user, read geometry and boundary condition data from files, invoke `fastlap`, and report on the results. The two most useful programs are the Fortran program called `driver.f` and the C program called `driver.c`.

These programs are provided:

- to guide you in tailoring your own Fortran or C code to call FastLap ;
- to allow you to experiment with setting FastLap flags without the uncertainties of linking to your own code;
- and to provide you with an easy means to test FastLap on your own input data.

The file of geometric input consists of lines of panel data for the boundary of the Laplace problem or field computation. Other elements in the `fastlap` argument list (described in §4.2) are provided by the driver depending on what type of example computation you elect to perform. The possibilities are a field computation, a Green formulation solution, a single-layer solution, a source formulation solution, a desingularized single-layer solution, and a desingularized source formulation solution. The two parameters which control the multipole acceleration `numMom`, the number of multipole coefficients and `numLev`, the number of levels in the hierarchical spatial decomposition, are provided by user in answers to queries in the Fortran program and in command line arguments in the C program.

To make the C program, type:

```
make driverc
```

This will create an executable `driverc`. To run this program, the command is, for example:

```
driverc -o2 -d4 -n1152 sphere.in
```

In this case, we are asking that the expansion order be 2, the depth of the cube hierarchy be 4, the number of panels be 1152, and the data for these panels will be found in `sphere.in`.

To make the Fortran program, type:

```
make driverf
```

This will create an executable `driverf`. To run this program, the command is:

```
driverf
```

When you run this Fortran version of the program you will be asked for the expansion order, the depth of the cube hierarchy, and the name of the panel data file.

As example data, there is a file provided in the distribution called `sphere.in`. This is the data for a sphere with unit radius translating in an infinite fluid. Neumann conditions are imposed on half of the sphere and Dirichlet conditions are imposed on the other half of the sphere. The sphere has been discretized into 1152 panels by equal subdivision of the polar [4] and azimuthal angles. The program which created this data is in the file `sphere.f`, so you may create other discretizations and boundary conditions for yourself.

The translating sphere is a useful example problem as the solution is known in closed form. For the unit sphere translating at unit velocity in an ideal fluid, in this case parallel to the x_3 axis and in the direction of its positive sense, the potential is known to be:

$$\psi(x) = -\frac{1}{2}\frac{x_3}{||x||^3}. \tag{2}$$

The program writes the exact values of the potential and the normal derivative of the potential into the data file `fastlap.in` regardless of what boundary conditions are selected (recall that boundary condition selection is controlled by `type`). Therefore, `driverc` and `driverf` have both the boundary conditions and the exact solution for this problem. This allows these programs to compare the solution computed by FastLap to the exact solution and report the magnitude of the average and maximum errors on the section of the sphere with Neumann boundary conditions, if you have specified one, and the section of the sphere with Dirichlet boundary conditions, if you have specified one.

`sphere` writes `fastlap.in` in a format which can be read by `driverc`. To create an input file which can be read by `driverf`, you must invoke the shell script `c2fdata` which is provided. This script uses two `sed` commands to create a file formated for `driverf` from one formated for `driverc`. For instance, to convert `sphere.in` to a file which can be read by `driverf`, say `sphere.dat`, the command is:

```
c2fdata sphere.in sphere.dat
```

Having made the Fortran driver as indicated above, the command to run it is:

```
driverf
```

Then the output written to `stdout` if the Green formulation option is selected will be as follows: [5]

[4] The polar angle is measured from the positive sense of the x_3 axis.

[5] A run of `driverc` will result in similar output.

```
xmeyer@flying-cloud.mit.edu> driverf


You can test the Green formulation, a single layer
formulation, the source formulation, or a field
computation.  If you use the input for the sphere
translating in an infinite fluid, there will be
error checking as the solution is known.  For
this case, the input file should be one written
by sphere.f and processed by c2fdata.
You may, of course, use your own input data.

Testing?
                   Field (0)
                   Green (1)
             Single Layer (2)
      Source Formulation (3)
   Repeated Single Layer (4)   1
You can run up to 8192 panels.

Select expansion order and tree depth:  2 4
Filename for input data?  sphere.dat
The header in this file is:
0  16x32 sphere
 Looking for          512 panels.

FastLap CONFIGURATION FLAGS:
  General Configuration
    NOWARN == OFF (warnings written on stdout)
    NOABRT == ON (slow fatal error traps disabled)
  Multipole Configuration
    DNTYPE == GRENGD (full Greengard dwnwd pass)
    MULTI == ON (include multipole part of P*q)
    RADINTER == ON (allow parent level interaction list entries)
    NNBRS == 2 (max distance to a nrst neighbor)
    ADAPT == ON (adaptive - no expansions in exact cubes)
    OPCNT == OFF (no P*q op count - iterate to convergence)
    MAXDEP == 20 (assume no more than 20 partitioning levels are needed)
  Linear System Solution Configuration
    ITRTYP == GMRES (generalized minimum residuals)
    PRECOND == OL (use overlapped preconditioner)
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.196034
```

```
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.20322
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.21587
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.231358
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.247029
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.260672
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.27065
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.275899
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.196034
FLW-placeq: oversized panel, cube length=0.124633 panel length=0.20322

FastLap ARG LIST SUMMARY:
  Number of singularities to process: 512
  Number of field or collocation points: 512
  Expansion order: 2
  Number of partitioning levels: 4
  Maximum allowed iterations: 32
  Convergence tolerance: 0.0001


MULTIPOLE SETUP SUMMARY:
  Level 0 cube extremal coordinates
    x: -0.980908 to 1.01322
    y: -0.980908 to 1.01322
    z: -0.98719 to 1.00693
  Level 4 (lowest level) cubes
    4096 total
    side length = 0.124633
  Maximum number of singularities in any = 9
  Maximum number of singularities treated exactly = 9 (limit = 9)
  No multipole expansions at level 4 (lowest)
  No multipole expansions at level 4 (lowest)
  Preconditioning matrix for inversion size = 34


GMRES RESIDUAL MONITOR:
  Iteration #1,  ||res|| = 0.049624
  Iteration #2,  ||res|| = 0.0324506
  Iteration #3,  ||res|| = 0.0125331
  Iteration #4,  ||res|| = 0.00615999
  Iteration #5,  ||res|| = 0.00222703
  Iteration #6,  ||res|| = 0.000680654
  Iteration #7,  ||res|| = 0.000205777
```

```
  Iteration #8,  ||res|| = 5.68166e-05
  Total GMRES iters = 8


TIME AND MEMORY USAGE SYNOPSIS
  Total time: 5.978
    Total setup time: 4.2173
      Data structure setup time: 0.334768
      Direct matrix setup time: 0.613904
      Multipole matrix setup time: 3.26862
      Initial misc. allocation time: 0
    Total iterative P*q = psi solve time: 1.7607
      P*q product time, direct part: 0.709552
      Total P*q time, multipole part: 0.847168
        Upward pass time: 0.010736
        Downward pass time: 0.0976
        Evaluation pass time: 0.738832
      Preconditioner solution time: 0.193248
      Iterative loop overhead time: 0.010736
  Total memory allocated: 5389 kilobytes (99.1% efficiency)
    Q2M  matrix memory allocated:      32 kilobytes
    Q2L  matrix memory allocated:     360 kilobytes
    Q2P  matrix memory allocated:    2931 kilobytes
    L2L  matrix memory allocated:       5 kilobytes
    M2M  matrix memory allocated:       6 kilobytes
    M2L  matrix memory allocated:      23 kilobytes
    M2P  matrix memory allocated:     219 kilobytes
    L2P  matrix memory allocated:      16 kilobytes
    Miscellaneous mem. allocated:    1793 kilobytes
    Total memory (check w/above):    5389 kilobytes


  8 iterations knocked down residual to:0.56816614E-04


Average absolute error on Dirichlet surface =   0.00380307
Maximum absolute error on Dirichlet surface =   0.02800870
xmeyer@flying-cloud.mit.edu>
```

Note that in this case there are panels which exceed the dimensions of finest-grain cubes, but accurate answers are still obtained. It is suggested that this same data file (`sphere.in` or `sphere.dat`) be used to run FastLap with other input parameters to see how run times and accuracy are affected.

There are variations on these computations which one can exercise by activating lines in `driver.f` by removing comment indicators. These include scrambling the ordering of the evaluation points, changing the extent of the desingularization, and moving the evaluation points in space.

8 The FastLap Distribution

A compressed tar file containing the files of the FastLap source code, the example driver and utilities source codes, the makefile, and this manual, is available via anonymous ftp at `rle-vlsi.mit.edu`. The archive is in `pub/fastlap` and its file name has the form:

```
fastlap-version-date.tar.Z
```

where `version` is the version number and `date` is the release date for the version. After retrieval, this archive may be unpacked by:

```
uncompress fastlap-version-date.tar.Z
tar xf fastlap-version-date.tar
```

The top-level directory is called `fastlap`. This directory contains the drivers, the utilities and a sample input file. `fastlap/` has two subdirectories: `fastlap/src/` which contains all the makefile, the C source and header files for FastLap ; and `fastlap/doc/` which contains the LaTeX, dvi, and postscript files for this manual.

Correspondence concerning FastLap should be sent to:

Dr. F. T. Korsmeyer
Research Laboratory *of* Electronics
Massachusetts Institute of Technology, Room 36-893
Cambridge, MA 02139 U.S.A.

Electronic mail may be sent to: `xmeyer@mit.edu`.

A Mathematical Background

In this section we review the theory for potential problems cast as boundary integral equations, but this review is quite terse and we encourage the reader to consult the references provided (for the details of multipole acceleration) and the standard texts (for the details of potential theory). We discuss the equation that results from the use of Green's Second Identity (or "Green's Theorem"). The extension to the single-layer, the source, and the desingularized formulations is straightforward.

FastLap is a procedure that can be used to solve Laplace problems cast as discretized boundary integral equations. FastLap requires only order N effort and storage, where N is the number of unknowns on the boundary. The reduction in cost compared to traditional order N^2 methods is achieved through never explicitly calculating all but an order N subset of the N^2 interactions implied by the linear system, while still providing the matrix-vector products that are required in the iterative solution of the problem [3].

If we consider the general, three-dimensional Laplace problem in the multiply-connected domain bounded by surfaces S_D, where Dirichlet boundary conditions are imposed ($\psi(x)$ is known), and S_N, where Neumann boundary conditions are imposed ($\psi_n(x)$ is known) (this includes the special cases of s trictly Neumann or Dirichlet boundary conditions) an integral equation can be derived via Green's theorem from which it follows that for each point x on a piecewise smooth surface $S = S_D \cup S_N$, the potential, $\psi(x)$, must satisfy

$$2\pi\psi(x) + \int_S \psi(x')G_n(x,x')da' - \int_S \psi_n(x')G(x,x')da' = 0. \tag{3}$$

where

$$G(x,x') = \frac{1}{\|x - x'\|} \tag{4}$$

and the subscript n denotes the operation $\hat{n} \cdot \vec{\nabla}$ in which $\hat{n}$ is the unit surface normal vector. Given (3), the surface potential can be determined uniquely if for each point x on S, the potential $\psi(x)$ and its normal derivative $\psi_n(x)$ are constrained to satisfy

$$\beta(x)\psi(x) + \gamma(x)\psi_n(x) = g(x). \tag{5}$$

An approach to numerically solving (3) and (5), referred to as a *panel method,* is to divide the surface S into M subsurfaces and then assume that ψ and ψ_n vary in some polynomial fashion over each subsurface, requiring a total of N coefficients. Insisting that this approximation satisfies (3) and (5) at a collection of N collocation points, denoted $\{x_i\}$, leads to a system of equations of the form

$$Dp - Pp_n = 0 \tag{6}$$

and

$$\Upsilon p + \Gamma p_n = g \tag{7}$$

where $p, p_n \in \Re^N$ are the vectors of coefficients of the polynomial approximations to ψ and ψ_n respectively, $\Upsilon, \Gamma \in \Re^{N \times N}$ are diagonal matrices whose diagonal elements are given by $\Upsilon_{jj} = \beta(x_j)$ and $\Gamma_{jj} = \gamma(x_j)$. The entries in $P, D \in \Re^{N \times N}$ are given by

$$P_{i,j} = \sum_{m=1}^{M} \int_{S_m} \frac{f(x')}{\|x' - x_i\|} da', \tag{8}$$

and

$$D_{i,j} = \sum_{m=1}^{M} \int_{S_m} f(x') \vec{\nabla} \frac{1}{\|x' - x_i\|} \cdot \hat{n} \; da', \tag{9}$$

where S_m is an approximate subsurface of S, defining the support of $f(x)$, a polynomial basis function. Upon specification of the boundary data, typically (but not necessarily) a collection of N values of $\psi(x)$ or $\psi_n(x)$, we have a linear system to solve that appears as:

$$Cq = \tilde{C}\tilde{q} \tag{10}$$

where $C_{i,j} = P_{i,j}, q_i = p_i, \tilde{C}_{i,j} = D_{i,j}$, and $\tilde{q}_i = p_{ni}$, if p_i is unknown; and $C_{i,j} = D_{i,j}, q_i = p_{ni}, \tilde{C}_{i,j} = P_{i,j}$, and $\tilde{q}_i = p_i$, if p_{ni} is unknown.

It is the vectors Cq and $\tilde{C}\tilde{q}$ that FastLap provides with order N cost. Apparently, the i^{th} element of Cq or $\tilde{C}\tilde{q}$ is the value of the potential at x_i due to the N singularities with influences described by $C_{i,j}$ and $\tilde{C}_{i,j}$ with strengths q and $\tilde{q}$, respectively.

The computation performed rapidly by FastLap is the evaluation the potential due to a collection of singularities of *known* strength. In the solution of a Laplace problem the strengths are known in the sense that they are either $\tilde{q}$, the boundary data, or q, an iterate in a solution algorithm in which we seek to minimize the residual in numerically evaluating equation (10).

References

[1] K. Nabors and J. White, "Fastcap: A multipole accelerated 3-D capacitance extraction program," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 1447–1459, November 1991.

[2] K. Nabors, F. T. Korsmeyer, F. T. Leighton and J. White, "Preconditioned, Adaptive, Multipole-Accelerated Iterative Methods for Three-Dimensional First-Kind Integral Equations of Potential Theory," *SIAM J. Sci. Comp.*, vol. 15, no. 3, pp. 713–735, May 1994.

[3] F. T. Korsmeyer, D. K. P. Yue, K. Nabors and J. White, "Multipole-Accelerated Preconditioned Iterative Methods for Three-Dimensional Potential Problems," *Proceedings of BEM 15* pp.517–527, Worcester, Massachusetts, August 1993.

[4] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comp.*, vol. 7, pp. 856–869, July 1986.

[5] J. N. Newman, "Distributions of sources and normal dipoles over a quadrilateral panel," *J. Eng. Math.*, no. 20, pp. 113–126, 1986.

[6] S. A. Vavasis, "Preconditioning for Boundary Integral Equations." *SIAM J. Matrix Anal. and App.*, vol. 13, no. 3, pp.905–925, July 1992.

[7] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comp. Phys.*, vol. 73, pp. 325–348, December 1987.

[8] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*. Cambridge, Massachusetts: M.I.T. Press, 1988.